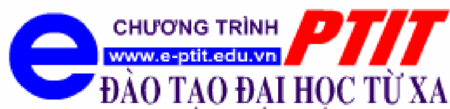
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



ĐIỆN TỬ SỐ

(Dùng cho sinh viên hệ đào tạo đại học từ xa)

Lưu hành nội bộ

HÀ NỘI - 2006

ĐIỆN TỬ SỐ

Biên soạn : ThS. TRẦN THỊ THÚY HÀ

LỜI GIỚI THIỆU

Cùng với sự tiến bộ của khoa học và công nghệ, các thiết bị điện tử đang và sẽ tiếp tục đợc ứng dụng ngày càng rộng rãi và mang lại hiệu quả cao trong hầu hết các lĩnh vực kinh tế kỹ thuật cũng như đời sống xã hội.

Việc xử lý tín hiệu trong các thiết bị điện tử hiện đại đều dựa trên cơ sở nguyên lý số. Bởi vậy việc hiểu sâu sắc về điện tử số là điều không thể thiếu được đối với kỹ sư điện tử hiện nay. Nhu cầu hiểu biết về kỹ thuật số không phải chỉ riêng đối với các kỹ sư điện tử mà còn đối với nhiều cán bộ kỹ thuật chuyên ngành khác có sử dụng các thiết bị điện tử.

Tài liệu này giới thiệu một cách hệ thống các phần tử cơ bản trong các mạch điện tử số kết hợp với các mạch điển hình, giải thích các khái niệm cơ bản về cổng điện tử số, các phương pháp phân tích và thiết kế mạch logic cơ bản.

Tài liệu bao gồm các kiến thức cơ bản về mạch cổng logic, cơ sở đại số logic, mạch logic tổ hợp, các trigơ, mạch logic tuần tự, các mạch phát xung và tạo dạng xung, các bộ nhớ thông dụng. Đặc biệt là trong tài liệu này có bổ xung thêm phần logic lập trình và ngôn ngữ mô tả phần cứng VHDL. Đây là ngôn ngữ phổ biến hiện nay dùng để tạo mô hình cho các hệ thống kỹ thuật số. Tất cả gồm 9 chương. Trước và sau mỗi chương đều có phần giới thiệu và phần tóm tắt để giúp người học dễ nắm bắt kiến thức hơn. Các câu hỏi ôn tập để người học kiểm tra mức độ nắm kiến thức sau khi học mỗi chương. Trên cơ sở các kiến thức căn bản, tài liệu đã cố gắng tiếp cận các vấn đề hiện đại, đồng thời liên hệ với thực tế kỹ thuật.

Tài liệu gồm có 9 chương được bố cục như sau:

Chương 1: Hệ đếm

Chương 2: Đại số Boole và các phương pháp biểu diễn hàm

Chương 3: Cổng logic TTL và CMOS

Chương 4: Mạch logic tổ hợp.

Chương 5: Mạch logic tuần tự.

Chương 6: Mạch phát xung và tạo dạng xung.

Chương 7: Bộ nhớ bán dẫn.

Chương 8: Logic lập trình.

Chương 9 : Ngôn ngữ mô tả phần cứng VHDL.

Do thời gian có hạn nên tài liệu này không tránh khỏi thiếu sót, rất mong người đọc góp ý. Các ý kiến xin gửi về Khoa Kỹ thuật Điện tử 1- Học viện Công nghệ Bưu chính viễn thông.

Xin trân trọng cảm ơn.

CHƯƠNG 1: HỆ ĐẾM

GIỚI THIỆU

Khi nói đến số đếm, người ta thường nghĩ ngay đến hệ thập phân với 10 chữ số được ký hiệu từ 0 đến 9. Máy tính hiện đại không sử dụng số thập phân, thay vào đó là số nhị phân với hai ký hiệu là 0 và 1. Khi biểu diễn các số nhị phân rất lớn, người ta thay nó bằng các số bát phân (Octal) và thập lục phân (HexaDecimal).

Đếm số lượng của các đại lượng là một nhu cầu của lao động, sản xuất. Ngừng một quá trình đếm, ta được một biểu diễn số. Các phương pháp đếm và biểu diễn số được gọi là **hệ đếm**. Hệ đếm không chỉ được dùng để biểu diễn số mà còn là công cụ xử lý.

Có rất nhiều hệ đếm, chẳng hạn như hệ La Mã, La Tinh ... Hệ đếm vừa có tính đa dạng vừa có tính đồng nhất và phổ biến. Mỗi hệ đếm có ưu điểm riêng của nó nên trong kĩ thuật số sẽ sử dụng một số hệ để bổ khuyết cho nhau.

Trong chương này không chỉ trình bày các hệ thập phân, hệ nhị phân, hệ bát phân, hệ thập lục phân và còn nghiên cứu cách chuyển đổi giữa các hệ đếm. Chương này cũng đề cập đến số nhị phân có dấu và khái niệm về dấu phẩy động.

NỘI DUNG

1.1. BIỂU DIỄN SỐ

Nguyên tắc chung của biểu diễn là dùng một số hữu hạn các ký hiệu ghép với nhau theo qui ước về vị trí. Các ký hiệu này thường được gọi là chữ số. Do đó, người ta còn gọi hệ đếm là hệ thống số. Số ký hiệu được dùng là cơ số của hệ ký hiệu là *r*. Giá trị biểu diễn của các chữ khác nhau được phân biệt thông qua trọng số của hệ. Trọng số của một hệ đếm bất kỳ sẽ bằng r^i, với *i* là một số nguyên dương hoặc âm.

Bảng 1.1 là liệt kê tên gọi, số ký hiệu và cơ số của một vài hệ đếm thông dụng.

Tên hệ đếm	Số ký hiệu	Cơ số (r)
Hệ nhị phân (Binary)	0, 1	2
Hệ bát phân (Octal)	0, 1, 2, 3, 4, 5, 6, 7	8
Hệ thập phân (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
Hệ thập lục phân (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	16

Bảng 1.1

Người ta cũng có thể gọi hệ đếm theo cơ số của chúng. Ví dụ: Hệ nhị phân = Hệ cơ số 2, Hệ thập phân = Hệ cơ số 10...

Dưới đây, ta sẽ trình bày tóm tắt một số hệ đếm thông dụng.

1.1.1 Hệ thập phân

Các ký hiệu của hệ như đã nêu ở bảng 1.1. Khi ghép các ký hiệu với nhau ta sẽ được một biểu diễn. Ví dụ: 1265,34 là biểu diễn số trong hệ thập phân:

$$1265.34 = 1\times10^3 + 2\times10^2 + 6\times10^1 + 5\times10^0 + 3\times10^{-1} + 4\times10^{-2}$$

Trong phân tích trên, 10^n **là trọng số của hệ**; các hệ số nhân chính là **ký hiệu** của hệ. Như vậy, *giá trị biểu diễn của một số trong hệ thập phân sẽ bằng tổng các tích của ký hiệu (có trong biểu diễn) với trọng số tương ứng*. Một cách tổng quát:

$$N_{10} = d_{n-1}\times10^{n-1} + \ldots + d_1\times10^1 + d_0\times10^0 + d_{-1}\times10^{-1} + \ldots + d_{-m}\times10^{-m}$$
$$= \sum_{n-1}^{-m} d_i \times 10^i$$

trong đó, N_{10} : biểu diễn bất kì theo hệ 10,

d : các hệ số nhân (ký hiệu bất kì của hệ),

n : số chữ số ở phần nguyên,

m : số chữ số ở phần phân số.

Ưu điểm của hệ thập phân là tính truyền thống đối với con người. Đây là hệ mà con người dễ nhận biết nhất. Ngoài ra, nhờ có nhiều ký hiệu nên khả năng biểu diễn của hệ rất lớn, cách biểu diễn gọn, tốn ít thời gian viết và đọc.

Nhược điểm chính của hệ là do có nhiều ký hiệu nên việc thể hiện bằng thiết bị kỹ thuật sẽ khó khăn và phức tạp.

Biểu diễn số tổng quát:

Với cơ số bất kì r và d bằng hệ số a tuỳ ý ta sẽ có công thức biểu diễn số chung cho tất cả các hệ đếm:

$$N = a_{n-1}\times r^{n-1} + \ldots + a_1\times r^1 + a_0\times r^0 + a_{-1}\times r^{-1} + \ldots + a_{-m}\times r^{-m}$$
$$= \sum_{n-1}^{-m} a_i \times r^i$$

Trong một số trường hợp, ta phải thêm chỉ số để tránh nhầm lẫn giữa biểu diễn của các hệ. Ví dụ: 36_{10}, 36_8, 36_{16}.

1.1.2 Hệ nhị phân

1.1.2.1. Tổ chức hệ nhị phân

Hệ nhị phân (Binary number system) còn gọi là hệ cơ số hai, gồm chỉ hai ký hiệu 0 và 1, cơ số của hệ là 2, trọng số của hệ là 2^n. Cách đếm trong hệ nhị phân cũng tương tự như hệ thập phân. Khởi đầu từ giá trị 0, sau đó ta cộng liên tiếp thêm 1 vào kết quả đếm lần trước. Nguyên tắc cộng nhị phân là : 0 + 0 = 0, 1 + 0 = 1, 1 + 1 = 10 ($10_2 = 2_{10}$).

Trong hệ nhị phân, mỗi chữ số chỉ lấy 2 giá trị hoặc 0 hoặc 1 và được gọi tắt là "bit". Như vậy, bit là số nhị phân 1 chữ số. Số bit tạo thành độ dài biểu diễn của một số nhị phân. Một số nhị phân có độ dài 8 bit được gọi 1 byte. Số nhị phân hai byte gọi là một từ (word). Bit tận cùng bên phải gọi là bit bé nhất (LSB – Least Significant Bit) và bit tận cùng bên trái gọi là bit lớn nhất (MSB - Most Significant Bit).

Biểu diễn nhị phân dạng tổng quát :

$$N_2 = b_{n-1}b_{n-2}....b_1b_0.b_{-1}b_{-2}....b_{-m}$$

Trong đó, b là hệ số nhân của hệ. Các chỉ số của hệ số đồng thời cũng bằng lũy thừa của trọng số tương ứng. Ví dụ :

1 1 0. 0 0 → số nhị phân phân số

2^2 2^1 2^0 2^{-1} 2^{-2} → trọng số tương ứng.

Các giá trị 2^{10} = 1024 được gọi là 1Kbit, 2^{20} = 1048576 - Mêga Bit ...

Ta có dạng tổng quát của biểu diễn nhị phân như sau:

$$N_2 = b_{n-1} \times 2^{n-1} + ... + b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + ... + b_{-m} \times 2^{-m}$$
$$= \sum_{n-1}^{-m} b_i \times 2^i$$

Trong đó, b là hệ số nhân lấy các giá trị 0 hoặc 1.

1.1.2.2. Các phép tính trong hệ nhị phân

a. Phép cộng

Qui tắc cộng hai số nhị phân 1 bit đã nêu ở trên.

b. Phép trừ

Qui tắc trừ hai bit nhị phân cho nhau như sau :

0 - 0 = 0 ; 1 - 1 = 0 ; 1 - 0 = 1 ; 10 - 1 = 1 (mượn 1)

Khi trừ nhiều bit nhị phân, nếu cần thiết ta mượn bit kế tiếp có trọng số cao hơn. Lần trừ kế tiếp lại phải trừ thêm 1.

c. Phép nhân

Qui tắc nhân hai bit nhị phân như sau:

0 x 0 = 0 , 0 x 1 = 0 , 1 x 0 = 0 , 1 x 1 = 1

Phép nhân hai số nhị phân cũng được thực hiện giống như trong hệ thập phân.

Chú ý : Phép nhân có thể thay bằng phép dịch và cộng liên tiếp.

d. Phép chia

Phép chia nhị phân cũng tương tự như phép chia hai số thập phân.

Ưu điểm chính của hệ nhị phân là chỉ có hai ký hiệu nên rất dễ thể hiện bằng các thiết bị cơ, điện. Các máy vi tính và các hệ thống số đều dựa trên cơ sở hoạt động nhị phân (2 trạng thái). Do

đó, hệ nhị phân được xem là ngôn ngữ của các mạch logic, các thiết bị tính toán hiện đại - ngôn ngữ máy.

Nhược điểm của hệ là biểu diễn dài, mất nhiều thời gian viết, đọc.

1.1.3 Hệ bát phân và thập lục phân

1.1.3.1 Hệ bát phân

1. Tổ chức của hệ : Nhằm khắc phục nhược điểm của hệ nhị phân, người ta thiết lập các hệ đếm có nhiều ký hiệu hơn, nhưng lại có quan hệ chuyển đổi được với hệ nhị phân. Một trong số đó là hệ bát phân (hay hệ Octal, hệ cơ số 8).

Hệ này gồm 8 ký hiệu : 0, 1, 2, 3, 4, 5, 6 và 7. Cơ số của hệ là 8. Việc lựa chọn cơ số 8 là xuất phát từ chỗ $8 = 2^3$. Do đó, mỗi chữ số bát phân có thể thay thế cho 3 bit nhị phân.

Dạng biểu diễn tổng quát của hệ bát phân như sau:

$$\begin{aligned} N_8 &= O_{n-1} \times 8^{n-1} + ... + O_0 \times 8^0 + O_{-1} \times 8^{-1} + ... + O_{-m} \times 8^{-m} \\ &= \sum_{n-1}^{-m} O_i \times 8^i \end{aligned}$$

Lưu ý rằng, hệ thập phân cũng đếm tương tự và có giải rộng hơn hệ bát phân, nhưng không thể tìm được quan hệ $10 = 2^n$ (với n nguyên).

2. Các phép tính trong hệ bát phân

a. Phép cộng

Phép cộng trong hệ bát phân được thực hiện tương tự như trong hệ thập phân. Tuy nhiên, khi kết quả của việc cộng hai hoặc nhiều chữ số cùng trọng số lớn hơn hoặc bằng 8 phải nhớ lên chữ số có trọng số lớn hơn kế tiếp.

b. Phép trừ

Phép trừ cũng được tiến hành như trong hệ thâp phân. Chú ý rằng khi mượn 1 ở chữ số có trọng số lớn hơn thì chỉ cần cộng thêm 8 chứ không phải cộng thêm 10.

Các phép tính trong hệ bát phân ít được sử dụng. Do đó, phép nhân và phép chia dành lại như một bài tập cho người học.

1.1.3.2 Hệ thập lục phân

1.Tổ chức của hệ

Hệ thập lục phân (hay hệ Hexadecimal, hệ cơ số 16). Hệ gồm 16 ký hiệu là 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Trong đó, $A = 10_{10}$, $B = 11_{10}$, $C = 12_{10}$, $D = 13_{10}$, $E = 14_{10}$, $F = 15_{10}$.

Cơ số của hệ là 16, xuất phát từ yếu tố $16 = 2^4$. Vậy, ta có thể dùng một từ nhị phân 4 bit (từ 0000 đến 1111) để biểu thị các ký hiệu thập lục phân. Dạng biểu diễn tổng quát:

$$N_{16} = H_{n-1}\times 16^{n-1} + ... + H_0 \times 16^0 + H_{-1}\times 16^{-1} + ... + H_{-m}\times 16^{-m}$$
$$= \sum_{n-1}^{-m} H_i \times 16^i$$

2. Các phép tính trong hệ cơ số 16

a. Phép cộng

Khi tổng hai chữ số lớn hơn 15, ta lấy tổng chia cho 16. Số dư được viết xuống chữ số tổng và số thương được nhớ lên chữ số kế tiếp. Nếu các chữ số là A, B, C, D, E, F thì trước hết, ta phải đổi chúng về giá trị thập phân tương ứng rồi mới cộng.

b. Phép trừ

Khi trừ một số bé hơn cho một số lớn hơn ta cũng mượn 1 ở cột kế tiếp bên trái, nghĩa là cộng thêm 16 rồi mới trừ.

c. Phép nhân

Muốn thực hiện phép nhân trong hệ 16 ta phải đổi các số trong mỗi thừa số về thập phân, nhân hai số với nhau. Sau đó, đổi kết quả về hệ 16.

1.2. CHUYỂN ĐỔI CƠ SỐ GIỮA CÁC HỆ ĐẾM

1.2.1. Chuyển đổi từ hệ cơ số 10 sang các hệ khác

Để thực hiện việc đổi một số thập phân đầy đủ sang các hệ khác ta phải chia ra hai phần: phần nguyên và phân số.

Đối với phần nguyên: ta chia liên tiếp phần nguyên của số thập phân cho cơ số của hệ cần chuyển đến, số dư sau mỗi lần chia viết đảo ngược trật tự là kết quả cần tìm. Phép chia dừng lại khi kết quả lần chia cuối cùng bằng 0.

Ví dụ: Đổi số 57_{10} sang số nhị phân.

Bước	chia	được	dư	
1	57/2	28	1	→ LSB
2	28/2	14	0	
3	14/2	7	0	
4	7/2	3	1	
5	3/2	1	1	
6	1/2	0	1	→ MSB

Viết đảo ngược trật tự, ta có : $57_{10} = 111001_2$

Đối với phần phân số : ta nhân liên tiếp phần phân số của số thập phân với cơ số của hệ cần chuyển đến, phần nguyên thu được sau mỗi lần nhân, viết tuần tự là kết quả cần tìm. Phép nhân dừng lại khi phần phân số triệt tiêu.

Ví dụ: Đổi số $57{,}34375_{10}$ sang số nhị phân.

Phần nguyên ta vừa thực hiện ở ví dụ a), do đó chỉ cần đổi phần phân số 0,375.

Bước	Nhân	Kết quả	Phần nguyên
1	0,375 x 2	0.75	0
2	0,75 x 2	1.5	1
3	0,5 x 2	1.0	1
4	0,0 x 2	0	0

Kết quả : $0,375_{10} = 0,0110_2$

Sử dụng phần nguyên đã có ở ví dụ 1) ta có : $57,375_{10} = 111001.0110_2$

1.2.2. Đổi một biểu diễn trong hệ bất kì sang hệ thập phân

Muốn thực hiện phép biến đổi, ta dùng công thức :

$$N_{10} = a_{n-1} \times r^{n-1} + ... + a_0 \times r^0 + a_{-1} \times r^{-1} + ... + a_{-m} \times r^{-m}$$

Thực hiện lấy tổng vế phải sẽ có kết quả cần tìm. Trong biểu thức trên, a_i và r là hệ số và cơ số hệ có biểu diễn.

1.2.3. Đổi các số từ hệ nhị phân sang hệ cơ số 8 và 16

Vì $8 = 2^3$ và $16 = 2^4$ nên ta chỉ cần dùng một số nhị phân 3 bit là đủ ghi 8 ký hiệu của hệ cơ số 8 và từ nhị phân 4 bit cho hệ cơ số 16.

Do đó, muốn đổi một số nhị phân sang hệ cơ số 8 và 16 ta chia số nhị phân cần đổi, kể từ dấu phân số sang trái và phải thành từng nhóm 3 bit hoặc 4 bit. Sau đó thay các nhóm bit đã phân bằng ký hiệu tương ứng của hệ cần đổi tới.

Ví dụ:

a. Đổi số $110111,0111_2$ sang số hệ cơ số 8

Tính từ dấu phân số, ta chia số này thành các nhóm 3 bit như sau :

110 111 , 011 100

↓ ↓ ↓ ↓

6 7 3 4

Kết quả: $110111,0111_2 = 67,34_8$. (Ta đã thêm 2 số 0 để tiện biến đổi).

b. Đổi số nhị phân $111110110,01101_2$ sang số hệ cơ số 16

Ta phân nhóm và thay thế như sau :

0001 1111 0110 0110 1000

↓ ↓ ↓ ↓ ↓

1 F 6 6 8

Kết quả: $111110110,01101_2 = 1F6,68_{16}$

1.3 SỐ NHỊ PHÂN CÓ DẤU

1.3.1 Biểu diễn số nhị phân có dấu

Có ba phương pháp thể hiện số nhị phân có dấu sau đây.

1. Sử dụng một bit dấu. Trong phương pháp này ta dùng một bit phụ, đứng trước các bit trị số để biểu diễn dấu, '0' chỉ dấu dương (+), '1' chỉ dấu âm (-).

2. Sử dụng phép bù 1. Giữ nguyên bit dấu và lấy bù 1 các bit trị số (bù 1 bằng đảo của các bit cần được lấy bù).

3. Sử dụng phép bù 2

Là phương pháp phổ biến nhất. Số dương thể hiện bằng số nhị phân không bù (bit dấu bằng 0), còn số âm được biểu diễn qua bù 2 (bit dấu bằng 1). Bù 2 bằng bù 1 cộng 1.

Có thể biểu diễn số âm theo phương pháp bù 2 xen kẽ: bắt đầu từ bit LSB, dịch về bên trái, giữ nguyên các bit cho đến gặp bit 1 đầu tiên và lấy bù các bit còn lại. Bit dấu giữ nguyên.

1.3.2 Các phép cộng và trừ số nhị phân có dấu

Như đã nói ở trên, phép bù 1 và bù 2 thường được áp dụng để thực hiện các phép tính nhị phân với số có dấu.

1. Biểu diễn theo bit dấu

a. Phép cộng

Hai số cùng dấu: cộng hai phần trị số với nhau, còn dấu là dấu chung.

Hai số khác dấu và **số âm** có trị số **nhỏ hơn:** cộng trị số của số dương với bù 1 của số âm. Bit tràn được cộng thêm vào kết quả trung gian. Dấu là dấu dương.

Hai số khác dấu và **số âm** có trị số **lớn hơn:** cộng trị số của số dương với bù 1 của số âm. Lấy bù 1 của tổng trung gian. Dấu là dấu âm.

b. Phép trừ. Nếu lưu ý rằng, - (-) = + thì trình tự thực hiện phép trừ trong trường hợp này cũng giống phép cộng.

2. Cộng và trừ các số theo biểu diễn bù 1

a. Cộng

Hai số dương: cộng như cộng nhị phân thông thường, kể cả bit dấu.

Hai số âm: biểu diễn chúng ở dạng bù 1 và cộng như cộng nhị phân, kể cả bit dấu. Bit tràn cộng vào kết quả. Chú ý, kết quả được viết dưới dạng bù 1.

Hai số khác dấu và số dương lớn hơn: cộng số dương với bù 1 của số âm. Bit tràn được cộng vào kết quả.

Hai số khác dấu và số âm lớn hơn: cộng số dương với bù 1 của số âm. Kết quả không có bit tràn và ở dạng bù 1.

b. Trừ

Để thực hiện phép trừ, ta lấy bù 1 của số trừ, sau đó thực hiện các bước như phép cộng.

3. Cộng và trừ nhị phân theo biểu diễn bù 2

a. Cộng

Hai số dương: cộng như cộng nhị phân thông thường. Kết quả là dương.

Hai số âm: lấy bù 2 cả hai số hạng và cộng, kết quả ở dạng bù 2.

Hai số khác dấu và số dương lớn hơn: lấy số dương cộng với bù 2 của số âm. Kết quả bao gồm cả bit dấu, bit tràn bỏ đi.

Hai số khác dấu và số âm lớn hơn: số dương được cộng với bù 2 của số âm, kết quả ở dạng bù 2 của số dương tương ứng. Bit dấu là 1.

b. Phép trừ

Phép trừ hai số có dấu là các trường hợp riêng của phép cộng. Ví dụ, khi lấy +9 trừ đi +6 là tương ứng với +9 cộng với -6.

1.4. DẤU PHẨY ĐỘNG

1.4.1 Biểu diễn theo dấu phẩy động

Gồm hai phần: số mũ E (phần đặc tính) và phần định trị M (trường phân số). E có thể có độ dài từ 5 đến 20 bit, M từ 8 đến 200 bit phụ thuộc vào từng ứng dụng và độ dài từ máy tính. Thông thường dùng 1 số bit để biểu diễn E và các bit còn lại cho M với điều kiện:

$$1/2 \le |M| \le 1$$

E và M có thể được biểu diễn ở dạng bù 2. Giá trị của chúng được hiệu chỉnh để đảm bảo mối quan hệ trên đây được gọi là chuẩn hóa.

1.4.2 Các phép tính với biểu diễn dấu phẩy động

Giống như các phép tính của hàm mũ. Giả sử có hai số theo dấu phẩy động đã chuẩn hóa: $X = 2^{E_x}(M_x)$ và $Y = 2^{E_y}(M_y)$ thì:

Tích: $Z = X.Y = 2^{E_x + E_y}(M_x.M_y) = 2^{E_Z} M_z$

Thương: $W = X/Y = 2^{E_x - E_y}(M_x / M_y) = 2^{E_w} M_w$

Muốn lấy tổng và hiệu, cần đưa các số hạng về cùng số mũ, sau đó số mũ của tổng và hiệu sẽ lấy số mũ chung, còn định trị của tổng và hiệu sẽ bằng tổng và hiệu các định trị.

TÓM TẮT

Trong chương này chúng ta giới thiệu về một số hệ đếm thường được sử dụng trong hệ thống số: hệ nhị phân, hệ bát phân, hệ thập lục phân. Và phương pháp chuyển đổi giữa các hệ đếm đó.

Ngoài ra còn giới thiệu các phép tính số học trong các hệ đó.

CÂU HỎI ÔN TẬP

1. Định nghĩa thế nào là bit, byte?
2. Đổi số nhị phân sau sang dạng bát phân: 0101 1111 0100 1110
   a. 57514
   b. 57515
   c. 57516
   d. 57517
3. Thực hiện phép tính hai số thập lục phân sau: $132,44_{16} + 215,02_{16}$.
   a. 347,46
   b. 357,46
   c. 347,56
   d. 357,67
4. Thực hiện phép cộng hai số có dấu sau theo phương pháp bù 1:

   $$0000\ 1101_2 + 1000\ 1011_2$$

   a. 0000 0101
   b. 0000 0100
   c. 0000 0011
   d. 0000 0010
5. Thực hiện phép cộng hai số có dấu sau theo phương pháp bù 2:

   $$0000\ 1101_2 - 1001\ 1000_2$$

   a. 1000 1110
   b. 1000 1011
   c. 1000 1100
   d. 1000 1110
6. Hai byte có bao nhiêu bit?
   a. 16
   b. 8
   c. 32
   d. 64

CHƯƠNG 2: ĐẠI SỐ BOOLE VÀ CÁC PHƯƠNG PHÁP BIỂU DIỄN HÀM

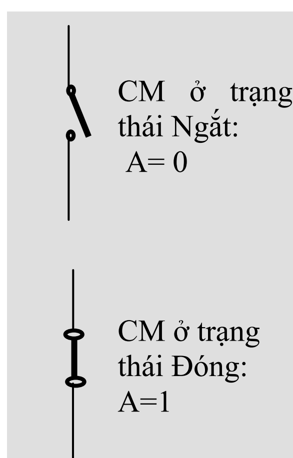
GIỚI THIỆU CHUNG

Trong mạch số, các tín hiệu thường cho ở hai mức điện áp, ví dụ 0 V và 5 V. Những linh kiện điện tử dùng trong mạch số làm việc ở một trong hai trạng thái, ví dụ transistor lưỡng cực làm việc ở chế độ khóa (tắt), hoặc thông..

Do vậy, để mô tả hoạt động của các mạch số, người ta dùng hệ nhị phân (Binary), hai trạng thái của các linh kiện trong mạch được mã hóa tương ứng thành 1 và 0.

Một bộ môn đại số được phát triển từ cuối thế kỷ 19 mang tên chính người sáng lập ra nó, đại số Boole, còn được gọi là đại số logic rất thích hợp cho việc mô tả mạch số. Đại số Boole là công cụ toán học quan trọng để thiết kế và phân tích mạch số. Các kỹ sư, các nhà chuyên môn trong lĩnh vực điện tử, tin học, thông tin, điều khiển.. đều cần phải nắm vững công cụ này để có thể đi sâu vào mọi lĩnh vực liên quan đến kỹ thuật số.

84 năm sau, đại số Boole đã được Shannon phát triển thành lý thuyết *chuyển mạch.* Nhờ các công trình của Shannon, về sau này, các nhà kỹ thuật đã dùng đại số Boole để phân tích và thiết kế các mạch vi tính. Trạng thái *"đúng", "sai"* trong bài toán logic được thay thế bằng trạng thái *"đóng", "ngắt"* của một ***chuyển mạch*** **(CM)**. Mối quan hệ nhân quả trong bài toán logic được thay bởi mối quan hệ giữa dòng điện trong mạch với trạng thái các CM gắn trên đoạn mạch ấy. Mối quan hệ này sẽ được thể hiện bằng một hàm toán học, có tên là ***hàm chuyển mạch.*** Khi đó, các trạng thái của CM : "đóng" = 1 và "ngắt" = 0. Hình 2-1 mô tả điều vừa nói. Ở đây, trạng thái của CM được kí hiệu bằng chữ cái A.



Về thực chất, hàm chuyển mạch là một trường hợp cụ thể của hàm logic. Do đó, đại số Boole ứng với trường hợp này cũng được gọi là đại số chuyển mạch. Mặc dù vậy, trong một số tài liệu người ta vẫn thường gọi nó là đại số logic hay đại số Boole.

Ngày nay, đại số Boole không chỉ giới hạn trong lĩnh vực kĩ thuật chuyển mạch mà còn là công cụ phân tích và thiết kế các mạch số, đặc biệt là lĩnh vực máy tính. Cấu kiện làm chuyển mạch được thay bằng Diode, Transistor, các mạch tích hợp, băng từ... Hoạt động của các cấu kiện này cũng được đặc trưng bằng hai trạng thái: thông hay tắt, dẫn điện hay không dẫn điện... Do đó, hai giá trị hệ nhị phân vẫn được dùng để mô tả trạng thái của chúng.

Đại số logic chỉ có 3 hàm cơ bản nhất, đó là hàm "Và", hàm "Hoặc" và hàm "Đảo". Đặc điểm nổi bật của đại số logic là cả hàm lẫn biến chỉ lấy hai giá trị hoặc 1 hoặc 0.

Trong chương này, ta sẽ đề cập đến các tiên đề, định lý, các cách biểu biễn hàm Boole và một số phương pháp rút gọn hàm. Ngoài ra, chương này cũng xét các loại cổng logic và các tham số chính của chúng.

NỘI DUNG

2.1 ĐẠI SỐ BOOLE

2.1.1. Các định lý cơ bản:

STT	Tên gọi	Dạng tích	Dạng tổng
1	Đồng nhất	$X.1 = X$	$X + 0 = X$
2	Phần tử 0, 1	$X.0 = 0$	$X + 1 = 1$
3	Bù	$X.\overline{X} = 0$	$X + \overline{X} = 1$
4	Bất biến	$X.X = X$	$X + X = X$
5	Hấp thụ	$X + X.Y = X$	$X.(X + Y) = X$
6	Phủ định đúp	$\overline{\overline{X}} = X$	
7	Định lý DeMorgan	$\overline{(X.Y.Z...)} = \overline{X} + \overline{Y} + \overline{Z} + ...$	$\overline{(X + Y + Z + ...)} = \overline{X}.\overline{Y}.\overline{Z}...$

Bảng 2.1. Một số định lý thông dụng trong đại số chuyển mạch

2.1.2 Các định luật cơ bản:

+ Hoán vị: $X.Y = Y.X$, $X + Y = Y + X$

+ Kết hợp: $X.(Y.Z) = (X.Y).Z$, $X + (Y + Z) = (X + Y) + Z$

+ Phân phối: $X.(Y + Z) = X.Y + X.Z$, $(X + Y).(X + Z) = X + Y.Z$

2.2 CÁC PHƯƠNG PHÁP BIỂU DIỄN HÀM BOOLE

Như đã nói ở trên, hàm logic được thể hiện bằng những biểu thức đại số như các môn toán học khác. Đây là phương pháp tổng quát nhất để biểu diễn hàm logic. Ngoài ra, một số phương pháp khác cũng được dùng để biểu diễn loại hàm này. Mỗi phương pháp đều có ưu điểm và ứng dụng riêng của nó. Dưới đây là nội dung của một số phương pháp thông dụng.

2.2.1 Bảng trạng thái

Liệt kê giá trị (trạng thái) mỗi biến theo từng cột và giá trị hàm theo một cột riêng (thường là bên phải bảng). Bảng trạng thái còn được gọi là ***bảng sự thật*** hay ***bảng chân lý***.

m	A	B	C	f
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	0
m_3	0	1	1	0
m_4	1	0	0	0
m_5	1	0	1	0
m_6	1	1	0	0
m_7	1	1	1	1

Bảng 2.2. Bảng trạng thái hàm 3 biến

Đối với hàm n biến sẽ có 2^n tổ hợp độc lập. Các tổ hợp này được kí hiệu bằng chữ m_i, với i = 0 đến 2^n -1 (xem bảng 2-2) và có tên gọi là các **hạng tích** hay còn gọi là **mintex**.

Vì mỗi hạng tích có thể lấy 2 giá trị là 0 hoặc 1, nên nếu có n biến thì số hàm mà bảng trạng thái có thể thiết lập được sẽ là: $N = 2^{2^n}$

2.2.2 Phương pháp bảng Các nô (Karnaugh)

Tổ chức của bảng Các nô: Các tổ hợp biến được viết theo một dòng (thường là phía trên) và một cột (thường là bên trái). Như vậy, một hàm logic có n biến sẽ có 2^n ô. Mỗi ô thể hiện một hạng tích hay một hạng tổng, các hạng tích trong hai *ô kế cận* chỉ khác nhau một biến.

Tính tuần hoàn của bảng Các nô: Không những các *ô kế cận khác nhau một biến* mà *các ô đầu dòng* và *cuối dòng*, *đầu cột* và *cuối cột* cũng chỉ *khác nhau một biến* (kể cả 4 góc vuông của bảng). Bởi vậy *các ô này cũng gọi là kế cận.*

Muốn thiết lập bảng Các nô của một hàm đã cho dưới dạng chuẩn tổng các tích, ta chỉ việc ghi giá trị 1 vào các ô ứng với hạng tích có mặt trong biểu diễn, các ô còn lại sẽ lấy giá trị 0 (theo định lý DeMorgan). Nếu hàm cho dưới dạng tích các tổng, cách làm cũng tương tự, nhưng các ô ứng với hạng tổng có trong biểu diễn lại lấy giá trị 0 và các ô khác lấy giá trị 1.

2.2.3 Phương pháp đại số

Có 2 dạng biểu diễn là dạng *tuyển* (*tổng các tích*) và dạng *hội* (*tích các tổng*).

+ Dạng tuyển: Mỗi số hạng là một *hạng tích* hay *mintex***,** thường kí hiệu bằng chữ **"m_i"**.

+ Dạng hội: Mỗi thừa số là *hạng tổng* hay *maxtex***,** thường được kí hiệu bằng chữ **"M_i".** Nếu trong tất cả mỗi hạng tích hay hạng tổng có đủ mặt các biến, thì dạng tổng các tích hay tích các tổng tương ứng được gọi là dạng *chuẩn*. Dạng chuẩn là duy nhất.

Tổng quát, hàm logic n biến có thể biểu diễn chỉ bằng một dạng tổng các tích:

$$f(X_{n-1},...,X_0) = \sum_{i=0}^{2^n-1} a_i m_i$$

hoặc bằng chỉ một dạng tích các tổng:

$$f\left(X_{n-1},...,X_0\right)=\prod_{i=0}^{2^n-1}\left(a_i+m_i\right)$$

Ở đây, a_i chỉ lấy hai giá trị 0 hoặc 1. Đối với một hàm thì mintex và maxtex là bù của nhau.

2.3 CÁC PHƯƠNG PHÁP RÚT GỌN HÀM

2.3.1. Phương pháp đại số

Dựa vào các định lý đã học để đưa biểu thức về dạng tối giản.

Ví dụ**:** Hãy đưa hàm logic về dạng tối giản:

$$f = AB + \overline{A}C + BC$$

Áp dụng định lý, $A+\overline{A}=1$, $X+XY=X$ ta có:

$$f = AB + \overline{A}C + BC\left(A+\overline{A}\right)$$
$$= AB + ABC + \overline{A}C + \overline{A}BC$$
$$= AB + \overline{A}C$$

Vậy nếu trong tổng các tích, xuất hiện một biến và đảo của biến đó trong hai số hạng khác nhau, các thừa số còn lại trong hai số hạng đó tạo thành thừa số của một số hạng thứ ba thì số hạng thứ ba đó là thừa và có thể bỏ đi.

2.3.2 Phương pháp bảng Các nô

Phương pháp này thường được dùng để rút gọn các hàm có số biến không vượt quá 5.

Các bước tối thiểu hóa:

1. Gộp các ô kế cận có giá trị '1' (hoặc '0') lại thành từng nhóm 2, 4, ..., 2^i ô. Số ô trong mỗi nhóm càng lớn kết quả thu được càng tối giản. Một ô có thể được gộp nhiều lần trong các nhóm khác nhau. Nếu gộp theo các ô có giá trị '0' ta sẽ thu được biểu thức bù của hàm.

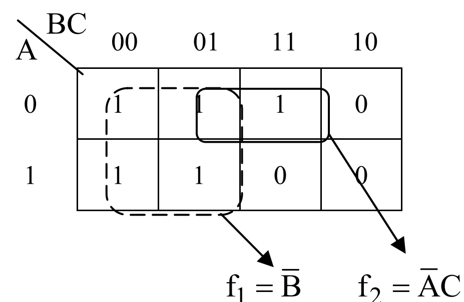
2. Thay mỗi nhóm bằng một hạng tích mới, trong đó giữ lại các biến giống nhau theo dòng và cột.

3. Cộng các hạng tích mới lại, ta có hàm đã tối giản.

Ví dụ: Hãy dùng bảng Các nô để giản ước hàm :

$$f\left(A,B,C\right)=\sum\left(1,\ 2,\ 3,\ 4,\ 5\right)$$

Lời giải:



Hình 2-2

+ Xây dựng bảng KN tương ứng với hàm đã cho.

+ Gộp các ô có giá trị 1 kế cận lại với nhau thành hai nhóm (hình 2-2)

Lời giải phải tìm :

$$f = f_1 + f_2 = \overline{B} + \overline{A}C$$

Nếu gộp các ô có giá trị 0 lại theo hai nhóm, ta thu được biểu thức hàm bù $\overline{f}$:

$$\overline{f} = AB + B\overline{C}$$

2.3.3. Phương pháp Quine Mc. Cluskey

Phương pháp này có thể tối thiểu hóa được hàm nhiều biến và có thể tiến hành công việc nhờ máy tính.

Các bước tối thiểu hóa:

1. Lập bảng liệt kê các hạng tích dưới dạng nhị phân theo từng nhóm với số bit 1 giống nhau và xếp chúng theo số bit 1 tăng dần.

2. Gộp 2 hạng tích của mỗi cặp nhóm chỉ khác nhau 1 bit để tạo các nhóm mới. Trong mỗi nhóm mới, giữ lại các biến giống nhau, biến bỏ đi thay bằng một dấu ngang (-).

Lặp lại cho đến khi trong các nhóm tạo thành không còn khả năng gộp nữa. Mỗi lần rút gọn, ta đánh dấu # vào các hạng ghép cặp được. Các hạng không đánh dấu trong mỗi lần rút gọn sẽ được tập hợp lại để lựa chọn biểu thức tối giản.

Ví dụ. Hãy tìm biểu thức tối giản cho hàm:

$$f(A, B, C, D) = \sum(10, 11, 12, 13, 14, 15)$$

Giải: Bước 1: Lập bảng (bảng 2.3a):

Bảng a		**Bảng b**	
Hạng tích đã sắp xếp	Nhị phân A B C D	Rút gọn lần đầu. *A B C D*	Rút gọn lần thứ 2. *A B C D*
10	1 0 1 0	1 0 1 - # (10,11)	1 1 - - (12,13,14,15)
<u>12</u>	<u>1 1 0 0</u>	1 - 1 0 # (10,14)	1 - 1 - (10,11,14,15)
11	1 0 1 1	1 1 0 - # (12,13)	
13	1 1 0 1	<u>1 1 - 0</u> # (12,14)	
<u>14</u>	<u>1 1 1 0</u>	1 - 1 1 # (11,15)	
15	1 1 1 1	1 1 - 1 # (13,15)	
		1 1 1 - # (14,15)	

Bảng 2.3

Bước 2: Thực hiện nhóm các hạng tích (bảng 2.3b).

Tiếp tục lập bảng lựa chọn để tìm hàm tối giản (Bảng 2.4):

A BCD	10	11	12	13	14	15
1 1 - -			x	x	x	x
1 - 1 -	x	x			x	x

Bảng 2.4

Từ bảng 2-4, ta nhận thấy rằng 4 cột có duy nhất một dấu "x" ứng với hai hạng 11-- và 1-1-. Do đó, biểu thức tối giản là :

$$f(A,B,C,D) = AB + AC$$

2.4 CỔNG LOGIC VÀ CÁC THAM SỐ CHÍNH

Cổng logic cơ sở là mạch điện thực hiện ba phép tính cơ bản trong đại số logic, vậy ta sẽ có ba loại cổng logic cơ sở là AND, OR và NOT.

2.4.1 Cổng logic cơ bản

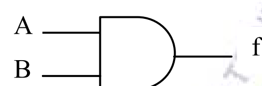
2.4.1.1 Cổng AND

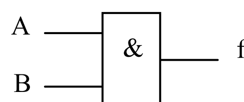
Cổng AND thực hiện hàm logic

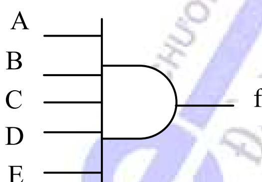
$$f = f(A,B) = A.B$$

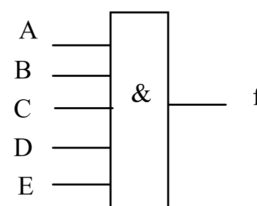
hoặc nhiều biến:

$$f(A,B,C,D,...) = A.B.C.D...$$









a) Theo tiêu chuẩn ANSI b) Theo tiêu chuẩn IEEE

Hình 2-4a,b. Ký hiệu của cổng AND.

Nguyên lý hoạt động của cổng AND:

Bảng trạng thái 2.5a,b là nguyên lí hoạt động của cổng AND (2 lối vào).

A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

a) Ghi theo giá trị logic

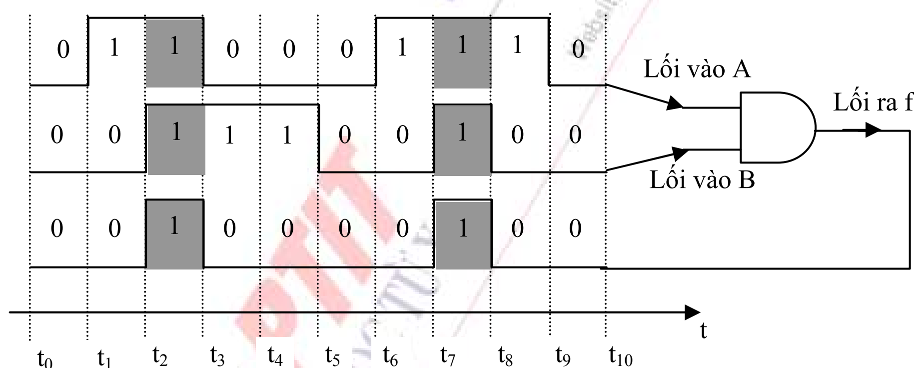
A	B	f
L	L	L
L	H	L
H	L	L
H	H	H

b) Ghi theo mức logic

Bảng 2.5a,b. Bảng trạng thái mô tả hoạt động của cổng AND 2 lối vào.

Theo qui ước, logic 1 được thay bằng mức điện thế cao, viết tắt là H (High) còn logic 0 được thay bằng mức điện thế thấp, viết tắt là L (Low) (bảng 2-5b). Cổng AND có n lối vào sẽ có 2^n hạng tích (dòng) trong bảng trạng thái.

Khi tác động tới lối vào các chuỗi xung số xác định, đầu ra cũng sẽ xuất hiện một chuỗi xung như chỉ hình 2-4. Đồ thị này thường được gọi là *đồ thị dạng xung, đồ thị dạng sóng hay đồ thị thời gian.*

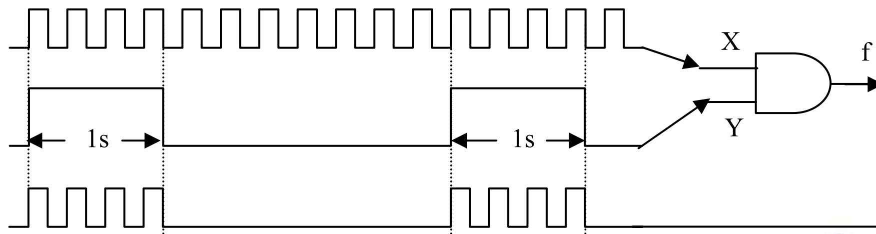


Hình 2-4. Đồ thị dạng xung vào, ra của cổng AND

Từ đồ thị, ta nhận thấy rằng, chỉ tại các thời điểm t_2 đến t_3 và t_7 đến t_8 trên cả hai lối vào đều có logic 1 nên lối ra cũng lấy logic 1. Ứng với các khoảng thời gian còn lại vì hoặc cả hai lối vào bằng 0, hoặc một trong hai lối vào bằng 0 nên lối ra lấy logic 0. Hoạt động của cổng AND nhiều lối vào cũng xảy ra tương tự.

Có thể giải thích dễ dàng một vài ứng dụng của cổng AND qua đồ thị dạng xung.

Ví dụ : Dùng cổng AND để tạo "cửa" thời gian. Trong ứng dụng này, trên hai lối vào của cổng AND được đưa tới 2 chuỗi tín hiệu số X, Y có tần số khác nhau. Giả sử tần số của X lớn hơn tần số của Y. Trên đầu ra cổng AND chỉ tồn tại tín hiệu X, gián đoạn theo từng chu kì của Y. Như vậy, chuỗi số Y chỉ giữ vai trò đóng, ngắt cổng AND và thường được gọi là tín hiệu "cửa". Hoạt động của mạch được mô tả bằng hình 2-5.



Hình 2-5. Mô hình dùng cổng AND để tạo "cửa" thời gian

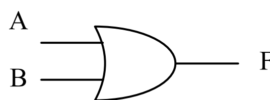
Tùy theo điều kiện cho trước, có thể ứng dụng mạch theo các mục đích khác nhau. Nếu đã biết độ rộng xung "cửa" Y (thường lấy bằng 1s) thì số xung xuất hiện đầu ra chính bằng tần số của X. Ngược lại, nếu tần số của X đã cho, chẳng hạn bằng 1 Hz (T_x = 1s) thì chỉ cần đếm số xung trên đầu ra ta có thể tính được độ rộng xung "cửa" Y. Đây chính là phương pháp đo tần số và thời gian được ứng dụng trong kĩ thuật hiện nay.

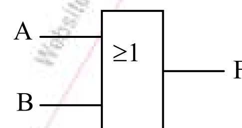
2.4.1.2 Cổng OR

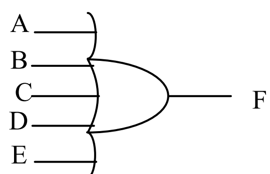
Cổng OR thực hiện hàm logic: $f(A,B) = A + B$

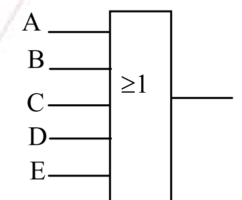
hoặc với hàm nhiều biến: $f(A,B,C,D...) = A + B + C + D + ...$

Ký hiệu của cổng OR được biểu diễn ở hình 2-6a, b.









a) Theo tiêu chuẩn ANSI b) Theo tiêu chuẩn IEEE

Hình 2-6 a, b. Ký hiệu của cổng OR.

Tương tự như cổng AND, nguyên lý hoạt động của cổng OR có thể được giải thích thông qua bảng trạng thái (Bảng 2.6a,b) và đồ thị dạng xung - hình 2-7.

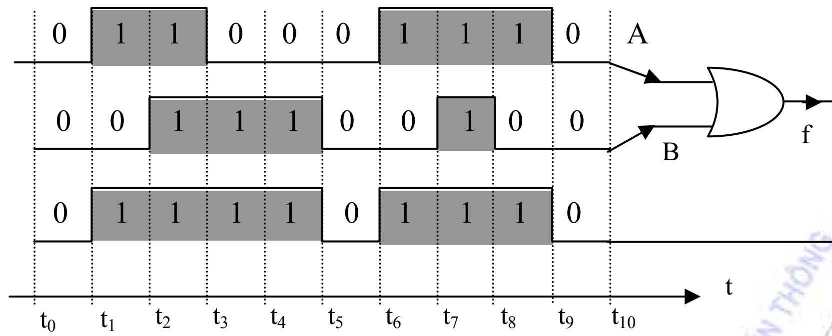
A	B	f
0	0	0
0	1	1
1	0	1
1	1	1

a) Theo giá trị logic

A	B	f
L	L	L
L	H	H
H	L	H
H	H	H

b) Theo mức điện thế

Bảng 2.6 a, b. Bảng trạng thái của cổng OR.



Hình 2-7. Đồ thị dạng xung của cổng OR.

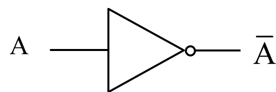
Một cổng OR có n lối vào sẽ có 2^n hạng tích trong bảng trạng thái của nó.

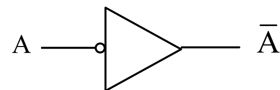
2.4.1.3. Cổng NOT

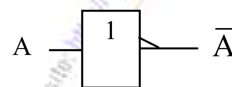
Cổng NOT thực hiện hàm logic:

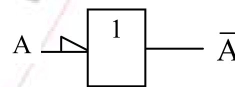
$$f = \overline{A}$$

Ký hiệu của cổng NOT được chỉ ra trên hình 2-8 a, b.









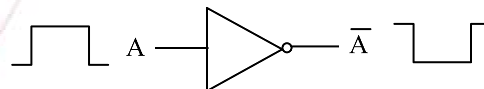
a) Theo tiêu chuẩn ANSI. b) Theo tiêu chuẩn IEEE.

Hình 2-8a,b. Ký hiệu của cổng NOT

Hoạt động của cổng NOT khá đơn giản, nếu lối vào:

$A = 0$ thì $\overline{A} = 1$,

nếu $A = 1$ thì $\overline{A} = 0$



Hình 2-9

Nguyên lý này được minh hoạ bằng đồ thị dạng xung ở hình 2-9.

Hoạt động của cổng NOT được tóm tắt ở bảng 2.7a,b.

A	f
0	1
1	0

a) Theo giá trị logic

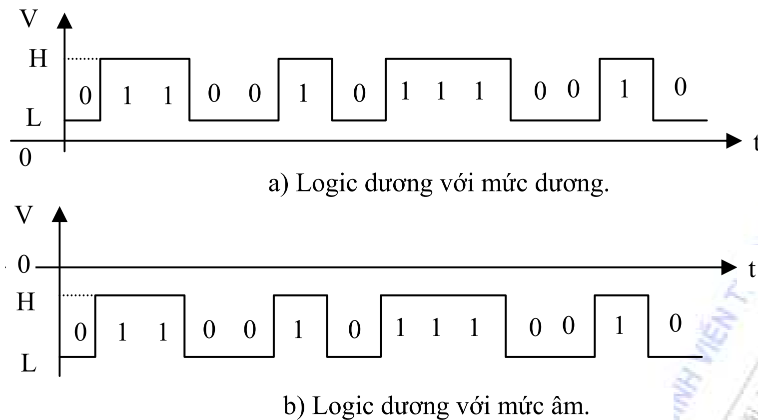
A	f
L	H
H	L

b) Theo mức logic

Bảng 2.7a, b. Bảng trạng thái của cổng NOT.

2.4.2 Logic dương và logic âm

Logic dương là logic có điện thế mức H luôn lớn hơn điện thế mức L (Hình 2-10).



a) Logic dương với mức dương.

b) Logic dương với mức âm.

Hình 2-10a,b. Đồ thị dạng xung của logic dương

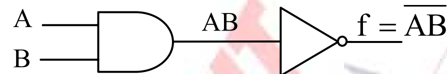
Logic âm thì ngược lại, logic 1 có điện thế thấp hơn mức 0. Khái niệm logic âm thường được dùng để biểu diễn trị các biến. Logic âm và mức âm của logic là hoàn toàn khác nhau.

2.4.3 Một số cổng ghép thông dụng

Khi ghép ba loại cổng logic cơ bản nhất sẽ thu được các mạch logic từ đơn giản đến phức tạp. Ở đây ta chỉ xét một vài mạch ghép đơn giản nhưng rất thông dụng.

2.4.3.1 Cổng NAND

Ghép nối tiếp một cổng AND với một cổng NOT ta được cổng NAND (Hình 2-11).



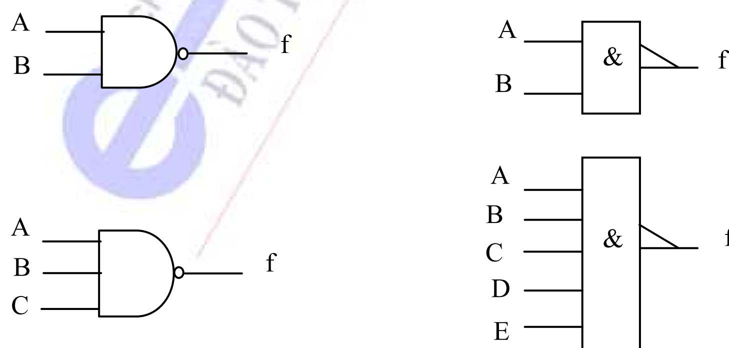
Hình 2-11. Sơ đồ cấu tạo cổng NAND

Hàm ra của cổng NAND 2 và nhiều biến vào như sau:

$$f = \overline{AB}$$

$$f = \overline{ABCD...}$$

Ký hiệu cổng NAND (hình 2-12a,b) và bảng trạng thái (bảng 2-8).



a) Theo tiêu chuẩn ANSI

b) Theo tiêu chuẩn IEEE

Hình 2-12a,b. Ký hiệu của cổng NAND

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

A	B	f
L	L	H
L	H	H
H	L	H
H	H	L

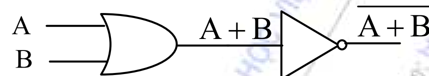
Bảng 2.8a,b. Bảng trạng thái của cổng NAND

2.4.3.2 Cổng NOR

Cổng NOR được thiết lập bằng cách nối tiếp một cổng OR với một cổng NOT.

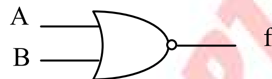
Từ hình 2-13 ta có thể viết được hàm ra của cổng NOR 2 và nhiều lối vào như sau:

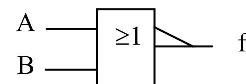
$f = \overline{A + B}$ hay $f = \overline{A + B + C + ...}$



Hình 2-13. Sơ đồ cấu tạo cổng NOR

Ký hiệu của cổng NOR 2 lối vào như chỉ ở hình 2-14a,b.





a) Theo tiêu chuẩn ANSI. b) Theo tiêu chuẩn IEEE.

Hình 2-14a, b. Ký hiệu cổng NOR 2 lối vào

Hoạt động của cổng NOR được giải thích bằng bảng trạng thái như chỉ ở bảng 2.9a,b.

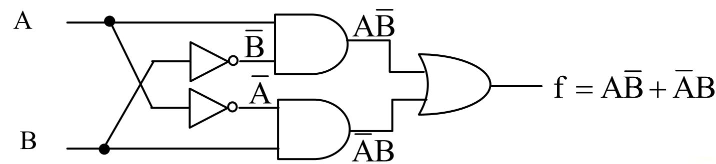
A	B	f
0	0	1
0	1	0
1	0	0
1	1	0

A	B	f
L	L	H
L	H	L
H	L	L
H	H	L

Bảng 2.9a, b. Bảng trạng thái của cổng NOR 2 lối vào.

2.4.3.3 Cổng khác dấu

Cổng *khác dấu* còn có một số tên gọi khác: cổng *Cộng Modul-2, cổng XOR*.

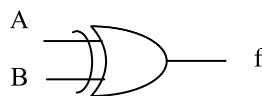


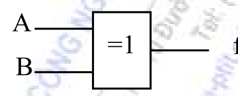
Hình 2-15. Sơ đồ của cổng XOR 2 lối vào

Từ hình 2-15, ta có biểu thức của hàm khác dấu 2 lối vào là:

$f = A\overline{B} + \overline{A}B$ hay theo qui ước $f = A \oplus B$

Ký hiệu của cổng XOR 2 lối vào như hình 2-16a, b.





a) Theo tiêu chuẩn ANSI b) Theo tiêu chuẩn IEEE

Hình 2-16a, b. Ký hiệu của cổng XOR 2 lối vào

Bảng trạng thái của cổng XOR hai lối vào được trình bày ở bảng 2.10a,b.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

A	B	F
L	L	L
L	H	H
H	L	H
H	H	L

Bảng 2-10a,b. Bảng trạng thái của cổng XOR 2 lối vào

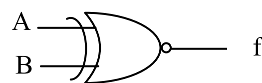
Hoạt động cổng XOR nhiều lối vào cũng tương tự như cổng 2 lối vào, nghĩa là nếu số bit 1 trên tất các các lối vào là một số lẻ, thì hàm ra lấy logic 1; ngược lại nếu tổng số bit 1 trên các lối vào là một số chẵn, thì hàm ra lấy logic 0. Có thể dùng cổng XOR 2 lối vào để thực hiện hàm XOR nhiều biến.

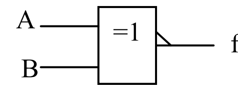
2.4.3.4 Cổng đồng dấu (XNOR)

Cổng XNOR thực hiện biểu thức logic sau:

$f = AB + \overline{A}\overline{B}$ hay $f = \overline{A \oplus B} = A \sim B$

Ký hiệu của cổng XNOR hai lối vào được trình bày ở hình 2-17.





a) Theo tiêu chuẩn ANSI b) Theo tiêu chuẩn IEEE

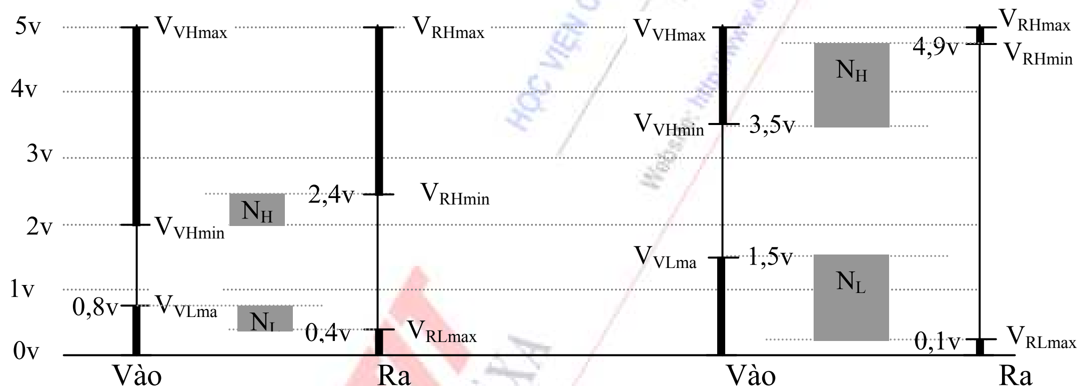
Hình 2-17. Ký hiệu của cổng XNOR 2 lối vào

Nếu tổng số bit 0 trên tất cả các lối vào là một số lẻ, thì hàm ra của XNOR sẽ lấy logic 1. Nếu tổng số bit 0 trên tất cả các lối vào là một số chẵn, thì hàm ra lại lấy logic 0.

XOR và XNOR là hai loại cổng có rất nhiều ứng dụng trong kỹ thuật số. Chúng là phần tử chính hợp thành bộ cộng, trừ , so sánh hai số nhị phân v.v...

2.4.4 Các tham số chính

2.4.4.1 Mức logic



a) Đối với họ TTL b) Đối với họ CMOS

Hình 2-19a, b. Mức logic của các họ cổng TTL và CMOS

Mức logic là mức điện thế trên đầu vào và đầu ra của cổng tương ứng với logic "1" và logic "0", nó phụ thuộc điện thế nguồn nuôi của cổng (V_{CC} đối với họ TTL (Transistor Transistor Logic) và V_{DD} đối với họ MOS (Metal Oxide Semiconductor)). Lưu ý rằng, nếu mức logic vào vượt quá điện thế nguồn nuôi có thể gây hư hỏng cho cổng.

Mức TTL

Mức TTL là một chuẩn quốc tế, trong đó qui định:

- Điện thế nguồn nuôi V_{CC} , V_{DD} bằng + 5 vôn hoặc bằng - 5,2 vôn;

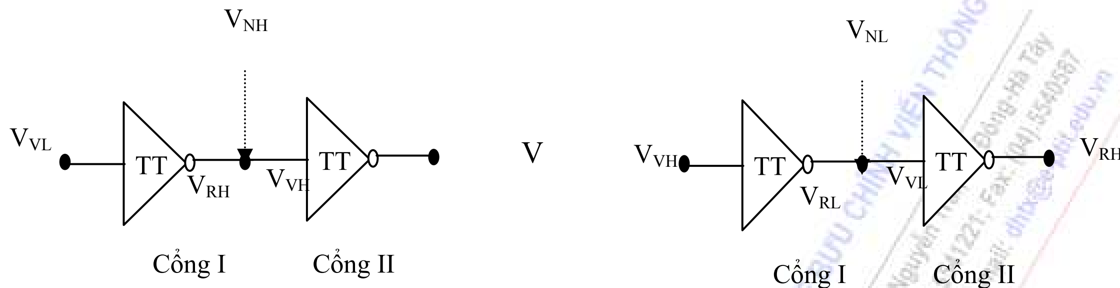
- Mức điện thế tương ứng với logic H và L trên đầu vào, đầu ra của cổng như chỉ ở hình 2-18a,b.

Nhận xét: + Mức vào ra đối với cổng TTL và CMOS (Complementary Metal Oxide Semiconductor) khác nhau rất nhiều;

+ Mức vào ra sẽ ảnh hưởng đến độ phòng vệ nhiễu của cổng.

2.4.4.2 Độ chống nhiễu

Độ chống nhiễu (hay độ phòng vệ nhiễu) là mức nhiễu lớn nhất tác động tới lối vào hoặc lối ra của cổng mà chưa làm thay đổi trạng thái vốn có của nó.



a) Tác động nhiễu khi mức ra cao b) Tác động nhiễu khi mức ra thấp

Hình 2-20a, b, Mô tả tác động nhiễu đến các cổng logic

Ảnh hưởng của nhiễu có thể phân ra hai trường hợp :

+ Nhiễu mức cao: đầu ra cổng I lấy logic H (hình 2-20a), tất nhiên, đầu ra cổng II là logic L, nếu các cổng vẫn hoạt động bình thường. Khi tính tới tác động của nhiễu, ta có:

$$V_{RH\,min} + V_{NH} \geq V_{VH\,min} \quad \Leftrightarrow V_{NH} \geq V_{VH\,min} - V_{RH\,min}$$

Với cổng TTL: $V_{NL} \geq 2V - 2,4V = -0,4V$

Với cổng CMOS: $V_{NL} \geq 3,5V - 4,9V = -1,4V$

+ Nhiễu mức thấp: đầu ra cổng I lấy logic L (hình 2-20b), tương tự ta có:

$$V_{RLmax} + V_{NL} \leq V_{VLmax} \quad \Leftrightarrow V_{NL} \leq V_{VLmax} - V_{RLmax}$$

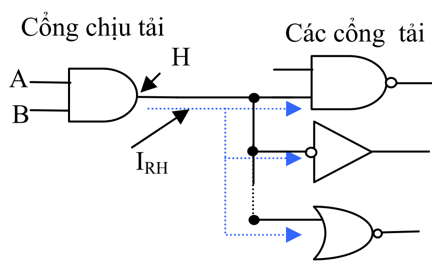
Với cổng TTL: $V_{NL} \leq 0,8V - 0,4V = 0,4V$

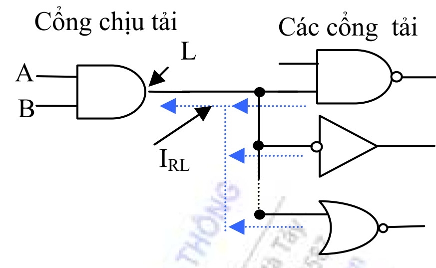
Với cổng CMOS: $V_{NL} \leq 1,5V - 0,1V = 1,4V$

2.4.4.3 Hệ số ghép tải K

Cho biết khả năng nối được bao nhiêu lối vào tới đầu ra của một cổng đã cho.

Hệ số ghép tải phụ thuộc dòng ra (hay dòng phun) của cổng chịu tải và dòng vào (hay dòng hút) của các cổng tải ở cả hai trạng thái H, L.

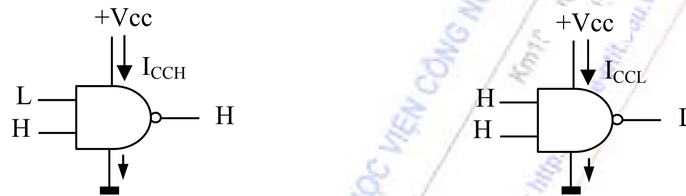




a) Mức ra của cổng chịu tải là H   b) Mức ra của cổng chịu tải là L

Hình 2-21a,b. Mô tả về hệ số ghép tải.

2.4.4.4. Công suất tiêu thụ



Hình 2-22. Hai trạng thái tiêu thụ dòng của cổng logic

I_{CCH} - Là dòng tiêu thụ khi đầu ra lấy mức H,

I_{CCL} - Là dòng tiêu thụ khi đầu ra lấy mức L.

Theo thống kê, tín hiệu số có tỷ lệ bit H / bit L khoảng 50%. Do đó, dòng tiêu thụ trung bình I_{CC} được tính theo công thức :

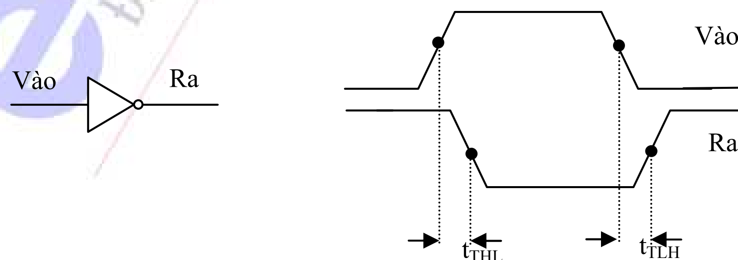
$$I_{CC} = (I_{CCH} + I_{CCL})/ 2$$

Công suất tiêu thụ trung bình của mỗi cổng sẽ là :

$$P_0 = I_{CC} \cdot V_{CC}$$

2.4.4.5. Trễ truyền lan

Tín hiệu đi qua một cổng phải mất một khoảng thời gian, được gọi là trễ truyền lan.



Hình 2-23. Minh hoạ trễ truyền lan của tín hiệu

Trễ truyền lan xảy ra tại cả hai sườn của xung ra. Nếu kí hiệu trễ truyền lan ứng với sườn trước là t_{THL} và sườn sau là t_{TLH} thì trễ truyền lan trung bình là:

$$t_{Ttb} = (t_{THL} + t_{TLH})/2$$

Thời gian trễ truyền lan hạn chế tần số công tác của cổng. Trễ càng lớn thì tần số công tác cực đại càng thấp.

TÓM TẮT

Trong chương 2 chúng ta giới thiệu về các phương pháp biểu diễn và rút gọn hàm Boole. Ngoài ra còn giới thiệu một số cổng logic thông dụng và các tham số chính của chúng.

CÂU HỎI ÔN TẬP

Bài 2.1 Rút gọn hàm sau theo phương pháp dùng bảng Karnaugh:

1. F (A, B, C) = Σ (0, 2, 4, 6,7).
   a. $AB + \overline{C}$
   b. $A\overline{B} + C$
   c. $AB + C$
   d. $\overline{A}B + C$
2. F (A, B, C, D) = Σ (0, 1, 8, 9, 10)
   a. $BC + D$
   b. $\overline{B}\overline{C} + A\overline{B}\overline{D}$
   c. $\overline{B}\overline{C} + ABD$
   d. $\overline{B}\overline{C} + \overline{A}\overline{B}D$

2.2 Rút gọn hàm sau theo phương pháp đại số

1. $\overline{CD + \overline{C}\,\overline{D}}\,.\,\overline{\overline{A}C + \overline{D}}$
   a. CD
   b. $C\overline{D}$
   c. $\overline{C}D$
   d. $\overline{C}\,\overline{D}$
2. $\overline{\overline{A}\,\overline{B}\,\overline{C}}\,.\,\overline{\overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{C}\,\overline{A}}$
   a. $AB + AC$
   b. $AB + AC + BC$
   c. $AC + BC$
   d. $AB + BC$

2.3 Rút gọn hàm sau theo phương pháp Quine-Mc.CLUSKEY:

F (A, B, C, D) = Σ (2, 3, 6, 7, 12, 13, 14, 15).

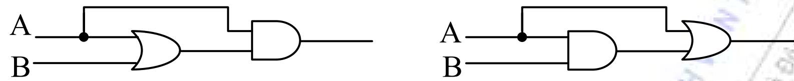
a. $AC + AB$

b. $\overline{A}C + AD$

c. $AC + \overline{A}B$

d. $\overline{A}C + AB$

2.4 Hai mạch điện ở hình dưới đây là tương đương



a. Do đều bằng A+B

b. Do đều bằng B

c. Do đều bằng AB

d. Do đều bằng A+AB

Bài 2.5 Phân tích ý nghĩa các tham số chính của các họ cổng logic.

Bài 2.6 Trình bày về độ phòng vệ nhiễu của các họ cổng logic? Tính độ phòng vệ nhiễu của một cổng logic họ TTL, biết V_{VL} = 0 V ÷ 0,8 V, V_{VH} = 2,0 V ÷ 5,0 V, V_{RL} = 0 V ÷ 0,4 V, V_{RH} = 2,4 V ÷ 5,0 V?

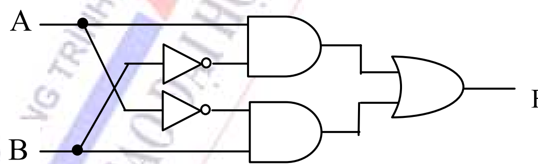
a. $V_{NH} = 0.4V,\ V_{NL} = -0.4$

b. $V_{NH} = -0.4V,\ V_{NL} = -0.4$

c. $V_{NH} = 0.4V,\ V_{NL} = 0.4$

d. $V_{NH} = -0.4V,\ V_{NL} = 0.4$

Bài 2.7 Cho mạch điện như hình 1. Biểu thức hàm ra là:



Hình 1

a. $AB + \overline{A}\overline{B}$

b. $\overline{A}B + \overline{A}\overline{B}$

c. $\overline{A}B + A\overline{B}$

d. $A\overline{B} + \overline{A}\overline{B}$

Bài 2.8 Phân tích ý nghĩa của việc tối ưu hoá mạch điện của các họ cổng logic? Cho ví dụ minh hoạ?

Bài 2.9 Chứng minh các đẳng thức:

a. $\overline{A \oplus B} = \overline{A}\ \overline{B} + AB$

b. $AB(A \oplus B \oplus C) = ABC$
c. $A \oplus B \oplus C = \overline{A} \oplus \overline{B} \oplus \overline{C}$

Bài 2.10 Liệt kê 3 phần tử logic cơ bản trong kỹ thuật số?

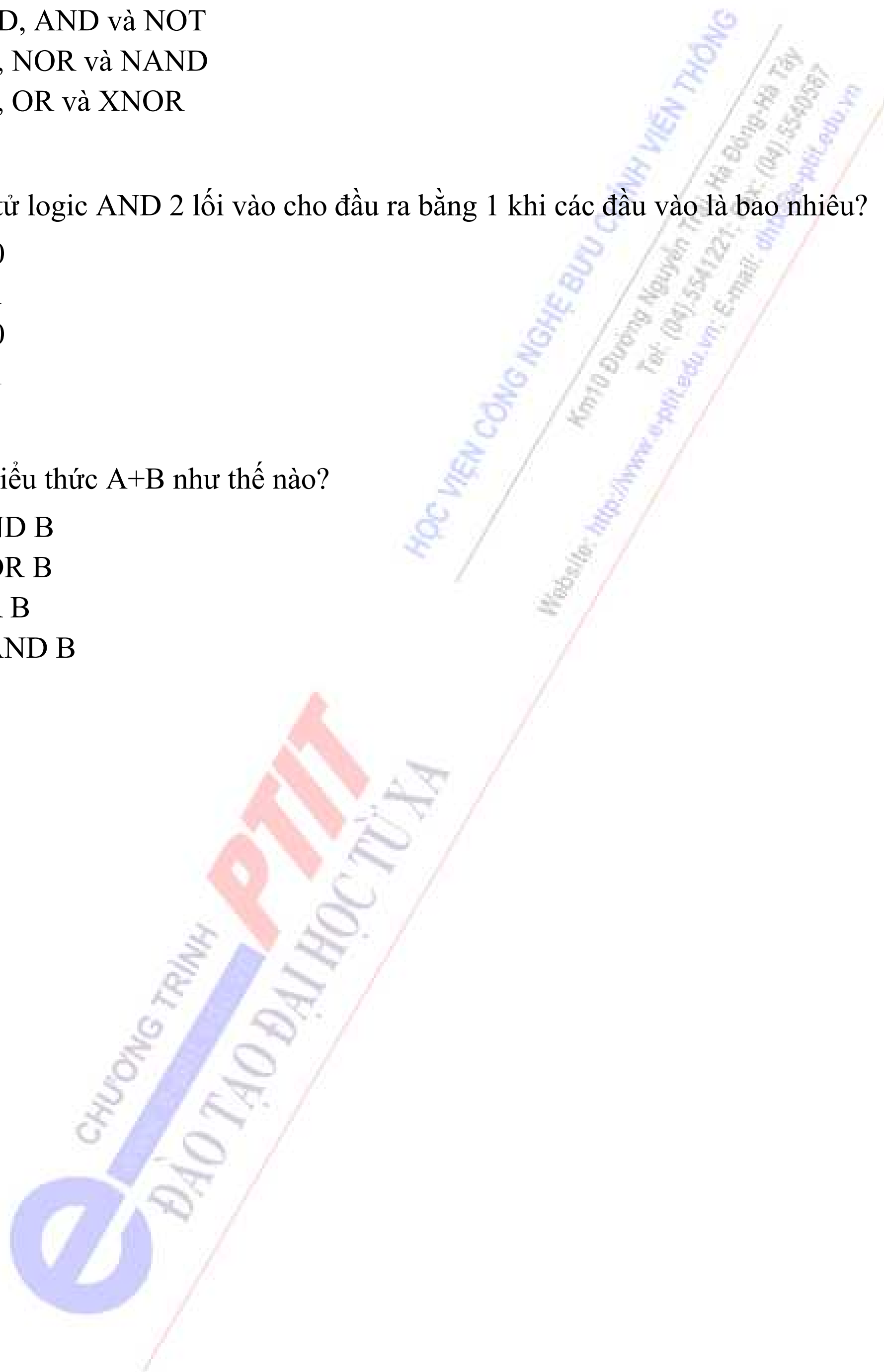
a. AND, OR và NOT
b. NAND, AND và NOT
c. AND, NOR và NAND
d. AND, OR và XNOR

Bài 2.11 Phần tử logic AND 2 lối vào cho đầu ra bằng 1 khi các đầu vào là bao nhiêu?

a. 0 và 0
b. 0 và 1
c. 1 và 0
d. 1 và 1

Bài 2.12 Đọc biểu thức A+B như thế nào?

a. A AND B
b. A XOR B
c. A OR B
d. A NAND B



CHƯƠNG 3: CỔNG LOGIC TTL VÀ CMOS

GIỚI THIỆU

Xét về mặt cơ bản thì có hai loại linh kiện bán dẫn đó là lưỡng cực và đơn cực. Dựa trên các linh kiện này, các mạch tích hợp được hình thành và có sẵn trên thị trường. Các chức năng kỹ thuật số khác nhau cũng được chế tạo trong nhiều dạng khác nhau bằng cách sử dụng công nghệ lưỡng cực và đơn cực. Một nhóm các IC tương thích với các mức logic giống nhau và các điện áp nguồn để thực hiện các chức năng logic đa dạng phải được chế tạo bằng cách sử dụng cấu hình mạch chuyên biệt được gọi là họ mạch logic.

Các yếu tố chính của một IC lưỡng cực là điện trở, điốt và các transistor. Có hai loại hoạt động cơ bản trong các mạch IC lưỡng cực:

- Bão hoà.
- Không bão hoà.

Trong mạch logic bão hoà, các transistor được vận hành trong vùng bão hoà, còn trong các mạch logic không bão hoà thì các transistor không làm việc tại vùng bão hoà.

Các họ mạch logic lưỡng cực được bão hoà là:

- Mạch logic Điện trở - Transistor (RTL).
- Mạch logic Điốt – Transistor (DTL).
- Mạch logic Transistor – Transistor (TTL).

Các họ mạch logic lưỡng cực không bão hòa là:

- Schottky TTL.
- Mạch logic ghép cực phát (ECL).

Các linh kiện MOS là các linh kiện đơn cực và chỉ có các MOSFET được vận hành trong các mạch logic MOS. Các họ mạch logic MOS là:

- PMOS.
- NMOS.
- CMOS

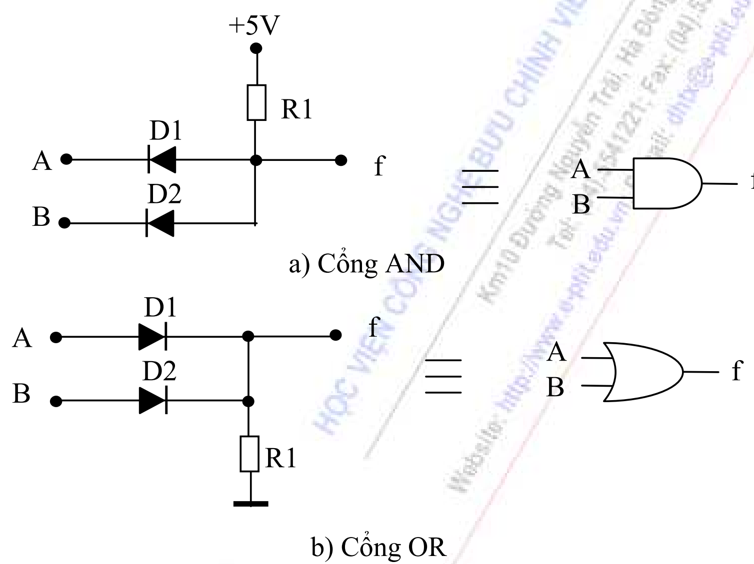
Trong chương 3 sẽ trình bày các họ cổng logic chủ yếu và được dùng phổ biến hiện nay. Phần cuối của chương trình bày một số mạch cho phép giao tiếp giữa các họ logic TTL và CMOS.

NỘI DUNG

3.1. CÁC HỌ CỔNG LOGIC

3.1.1. Họ DDL

DDL (Diode Diode Logic) là họ cổng logic do các diode bán dẫn tạo thành. Hình 3-1a,b là sơ đồ cổng AND, OR 2 lối vào họ DDL.



Hình 3-1. Mạch điện cổng AND và OR họ DDL.

Bảng trạng thái sau thể hiện nguyên lý hoạt động của mạch thông qua mức điện áp vào/ra của các cổng AND và OR họ DDL

	AND			OR	
A (V)	B (V)	F (V)	A (V)	B (V)	F (V)
0	**0**	**0,7**	**0**	**0**	**0**
0	**3**	**0,7**	**0**	**5**	**4,3**
3	**0**	**0,7**	**5**	**0**	**4,3**
3	**3**	**4,7**	**5**	**5**	**4,3**

Bảng 3-1. Bảng trạng thái của cổng AND và OR họ DDL

Ưu điểm của họ DDL:

- Mạch điện đơn giản, dễ tạo ra các cổng AND, OR nhiều lối vào. Ưu điểm này cho phép xây dựng các ma trận diode với nhiều ứng dụng khác nhau;
- Tần số công tác có thể đạt cao bằng cách chọn các diode chuyển mạch nhanh;
- Công suất tiêu thụ nhỏ.

Nhược điểm :

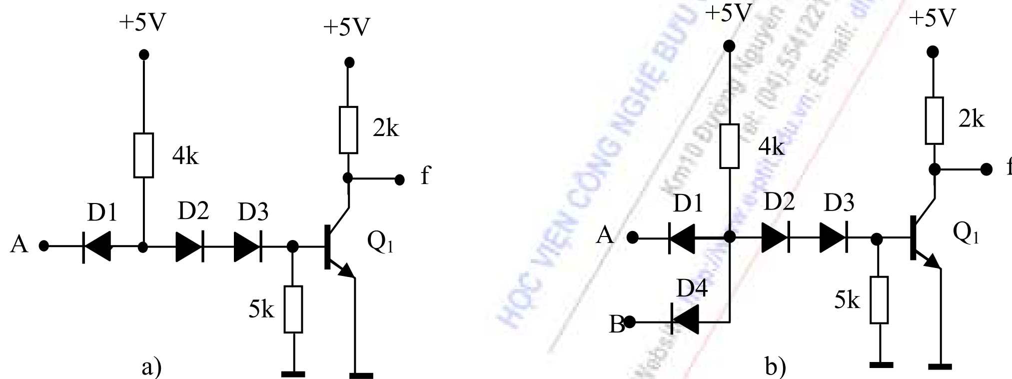
– Độ phòng vệ nhiễu thấp (V_{RL} lớn) ;

– Hệ số ghép tải nhỏ.

Để cải thiện độ phòng vệ nhiễu ta có thể ghép nối tiếp ở mạch ra một diode. Tuy nhiên, khi đó V_{RH} cũng bị sụt đi 0,6V.

3.1.2. Họ DTL

Để thực hiện chức năng đảo, ta có thể đấu nối tiếp với các cổng DDL một transistor công tác ở chế độ khoá. Mạch cổng như thế được gọi là họ DTL (Diode Transistor Logic). Ví dụ, hình 3-2a, b là các cổng NOT, NAND thuộc họ này.



Hình 3-2. Sơ đồ mạch điện của họ cổng TDL.

Trong hai trường hợp trên, nhờ các diode D2, D3 độ chống nhiễu trên lối vào của Q_1 được cải thiện. Mức logic thấp tại lối ra f giảm xuống khoảng 0,2 V (bằng thế bão hoà U_{CE} của Q_1). Do I_{RHmax} và I_{RLmax} của bán dẫn có thể lớn hơn nhiều so với diode nên hệ số ghép tải của cổng cũng tăng lên.

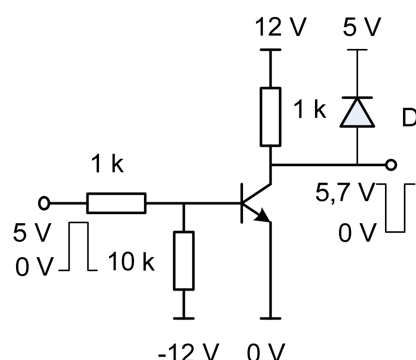
Bằng cách tương tự, ta có thể thiết lập cổng NOR hoặc các cổng liên hợp phức tạp hơn. Vì tải của các cổng là điện trở nên hệ số ghép tải (đặc biệt đối với N_H) còn bị hạn chế, mặt khác trễ truyền lan của họ cổng này còn lớn. Những tồn tại trên sẽ được khắc phục từng phần ở các họ cổng sau.

3.1.3. Họ RTL

Họ RTL (Resistor Transistor Logic) là các cổng logic được cấu tạo bởi các điện trở và transistor. Hình 3-3 là sơ đồ của một mạch NOT họ RTL.

Khi điện áp lối vào là 0 V, điện áp trên base của transistor sẽ âm nên transistor cấm như vậy lối ra trên collector của transistor sẽ ở mức cao. Do lối ra này được nối lên nguồn +5 V thông qua diode D nên giá trị điện áp lối ra lúc này khoảng 5,7 V, nhận mức logic cao. Khi điện áp lối vào là 5 V do hai điện trở lối vào có giá trị lần lượt là 1 k và 10 k, nên điện áp tại base sẽ đủ lớn để làm transistor thông làm cho điện áp lối ra là 0 V. Như vậy logic lối ra sẽ là đảo của logic của tín hiệu lối vào.

Tương tự như mạch hình 3-3, nếu một điện trở được nối thêm ở lối vào như hình 3-4 sau mạch sẽ trở thành mạch NOR họ RTL.

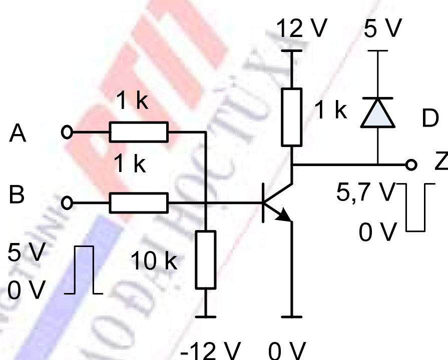


Hình 3-3. Cổng NOT họ RTL

Bảng 3-2 thể hiện quan hệ điện áp của cổng NOR họ RTL, chỉ khi cả hai lối vào A và B cùng ở giá trị 0 V thì transistor mới cấm và lối ra nhận logic cao. Các trường hợp khác đều dẫn đến transistor thông và làm giá trị logic lối ra ở mức thấp.

A (V)	**B (V)**	**F (V)**
0	**0**	**5,7**
0	**5**	**0**
5	**0**	**0**
5	**5**	**0**

Bảng 3-2. Bảng trạng thái của cổng NOR họ RTL

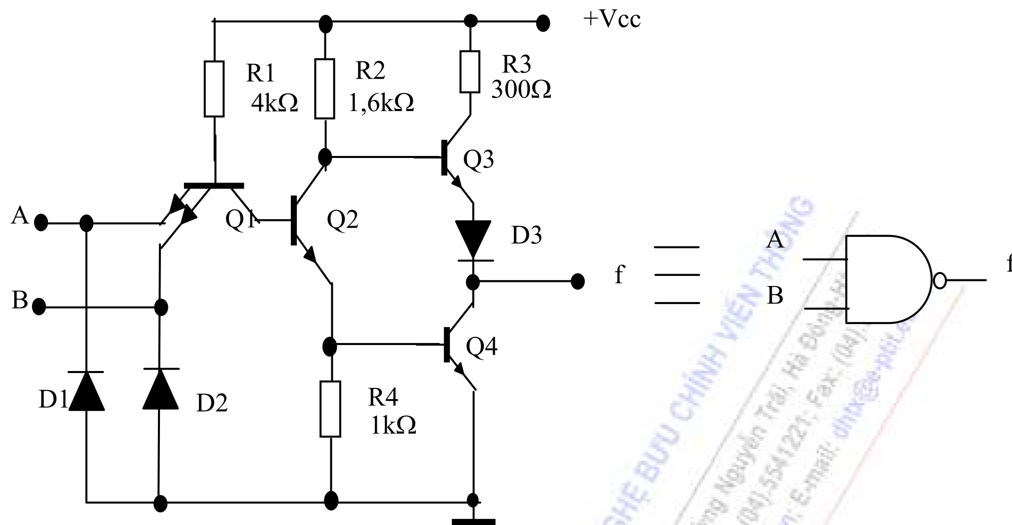


Hình 3.4. Cổng NOR họ RTL

3.1.4. Họ TTL

Do hạn chế về tốc độ, họ DTL đã trở nên lạc hậu và bị thay thế hoàn toàn bởi họ mạch TTL. Hạn chế tốc độ của DTL được giải quyết bằng cách thay các điốt đầu vào thành transistor đa lớp tiếp giáp BE.

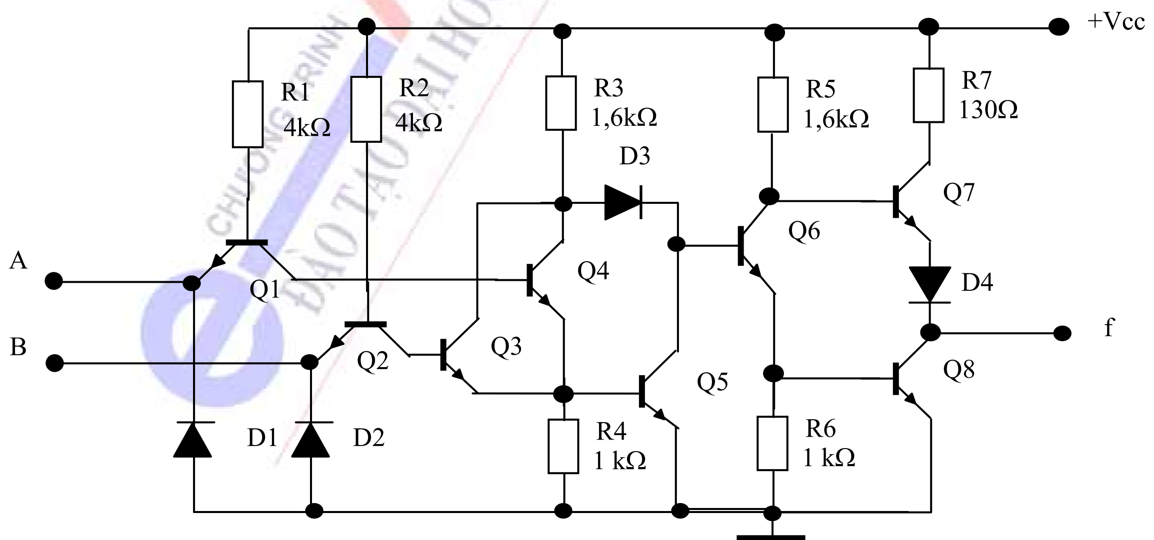
a. Cổng NAND TTL



Hình 3-5. Sơ đồ mạch điện một cổng NAND 2 lối vào.

Hình 3-5 là sơ đồ nguyên lý của mạch NAND TTL. Nó có thể được chia ra thành 3 phần. Transistor Q_1, trở R_1 và các diode D_1, D_2 tạo thành mạch đầu vào, mạch này thực hiện chức năng NAND. Transistor Q_2, các trở R_2, R_4 tạo thành mạch giữa Q_3, Q_4, R_3 và diode D_3 tạo thành mạch lối ra như phân tích ở trên.

Khi bất kỳ một lối vào ở mức thấp thì Q_1 đều trở thành thông bão hoà, do đó, Q_2 và Q_4 đóng, còn Q_3 thông nên đầu ra của mạch sẽ ở mức cao. Lối ra sẽ chỉ xuống mức thấp khi tất cả các lối vào đều ở mức logic cao và làm transistor Q_1 cấm. Diode D_3 được sử dụng như mạch dịch mức điện áp, nó có tác dụng làm cho Q_3 cấm hoàn toàn khi Q_2 và Q_4 thông. Diode này nhiều khi còn được mắc vào mạch giữa collector Q_2 và base của Q_3.



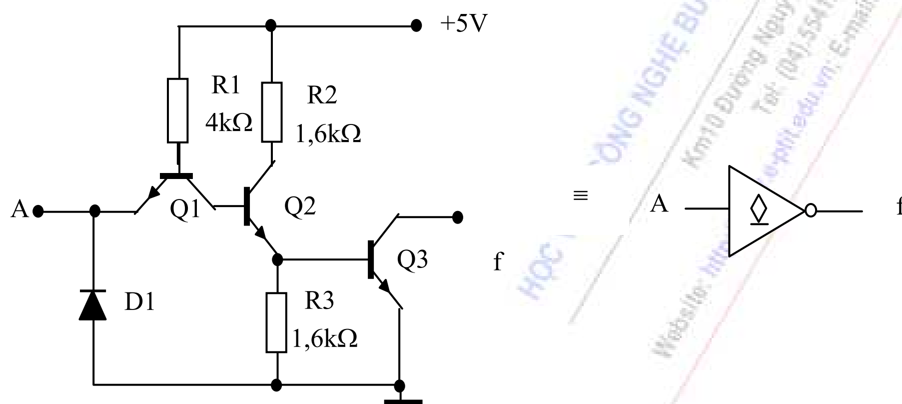
Hình 3-6. Sơ đồ mạch điện của một cổng OR 2 lối vào.

b. Cổng OR TTL

Hình 3-6 là sơ đồ của một cổng OR họ TTL tiêu chuẩn hai lối vào. Trong trường hợp này, mạch vào sử dụng các bán dẫn đơn. Tuy nhiên, nguyên lý hoạt động của mạch vào này cũng giống với cổng NAND hình 3-5.

c. Cổng collector để hở

Nhược điểm của họ cổng TTL có mạch ra khép kín là hệ số tải đầu ra không thể thay đổi, nên nhiều khi gây khó khăn trong việc kết nối với đầu vào của các mạch điện tử tầng sau. Cổng logic collector để hở khắc phục được nhược điểm này. Hình 3-7 là sơ đồ của một cổng TTL đảo collector hở tiêu chuẩn. Muốn đưa cổng vào hoạt động, cần đấu thêm trở gánh ngoài, từ cực collector đến +Vcc.



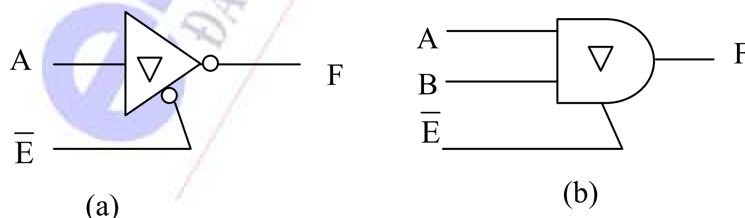
Hình 3-7. Mạch điện của một cổng NOT collector hở.

Một nhược điểm của cổng logic collector hở là tần số hoạt động của mạch sẽ giảm xuống do phải sử dụng điện trở gánh ngoài.

d. Cổng TTL 3 trạng thái

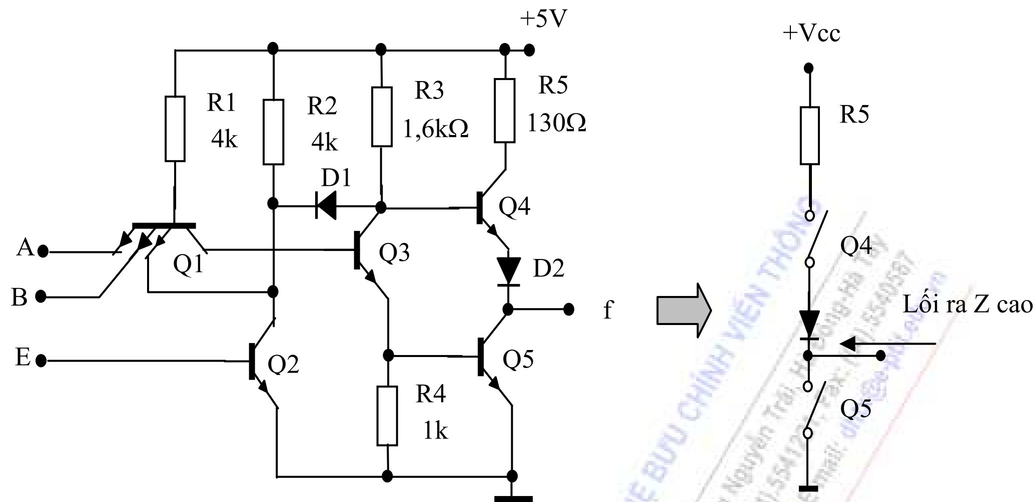
Một cổng logic, ngoài hai trạng thái cao và thấp tại đầu ra của nó còn có một trạng thái trung gian được gọi là cổng ba trạng thái. Trạng thái trung gian này còn có tên là trạng thái đầu ra có trở kháng Z cao hay trạng thái treo. Cổng có ký hiệu như chỉ ở hình 3-8.

Tương tự như cổng collector hở, các họ cổng logic đều có cổng 3 trạng thái. Hình 3-8 là một ví dụ về mạch điện của cổng NAND ba trạng thái họ TTL tiêu chuẩn .



Hình 3-8. Ký hiệu của cổng ba trạng thái : (a) cổng NOT; (b) cổng AND.

Hoạt động của cổng NAND 3 trạng thái được giải thích bằng bảng trạng thái 3-3. Khi trên lối vào E có mức logic thấp, cổng hoạt động như một cổng NAND. Trên lối ra *f* sẽ tồn tại hai trạng thái cao và thấp như thường lệ.



Hình 3-9. Mạch điện cổng NAND 3 trạng thái và sơ đồ tương đương của nó.

E	A	B	f
L	L	L	H
L	L	H	H
L	H	L	H
L	H	H	L
H	x	x	-
H	x	x	-
H	x	x	-
H	x	x	-

Bảng 3-3. Bảng trạng thái của cổng 3 trạng thái.

Ngược lại, khi trên lối vào E ở mức cao thì bất luận trên hai lối vào A, B có giá trị logic nào (dấu x trong bảng trạng thái mang ý nghĩa tuỳ chọn) lối ra f luôn ở trạng thái treo, hay thả nổi. Trạng thái này tương đương với trạng thái đầu ra không được nối tới một điểm nào trong mạch. Ứng với trạng thái này, trở kháng Z trên đầu ra của cổng, nhìn từ phía tải vào sẽ rất lớn. Theo sơ đồ tương đương, lúc này cả Q_4, Q_5 đều khoá. Lối ra f dường như bị treo trong mạch. Do đó, trạng thái này còn được gọi là trạng thái treo.

Trong kỹ thuật số, cổng ba trạng thái thường được dùng làm các bộ đệm đầu ra, khoá điều khiển hướng dữ liệu ...

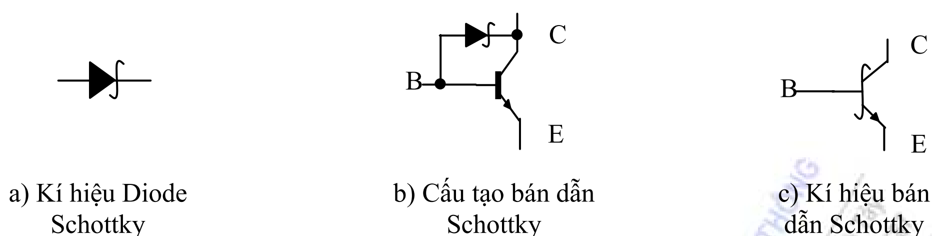
e. Họ TTL có diode Schottky (TTL + S)

Cổng TTL tiêu chuẩn có nhược điểm chung là thời gian trễ truyền lan lớn. Nguyên nhân của nhược điểm này là do tất cả bán dẫn trong mạch đều công tác ở chế độ bão hoà. Một trong những biện pháp giảm nhỏ trễ truyền lan là sử dụng diode Schottky để chống hiện tượng bão hoà này.

– **Diode và bán dẫn Schottky**

Cấu tạo của diode Schottky cũng giống như diode Silic. Nhờ việc chèn thêm một lớp oxit kim loại vào giữa tiếp giáp p-n mà điện thế phân cực của nó là 0,4 Vdc (thấp hơn 0,6 vôn đối với diode Silic và cao hơn 0,2 với diode Ge).

Ký hiệu của diode và bán dẫn Schottky cho ở hình 3-10.



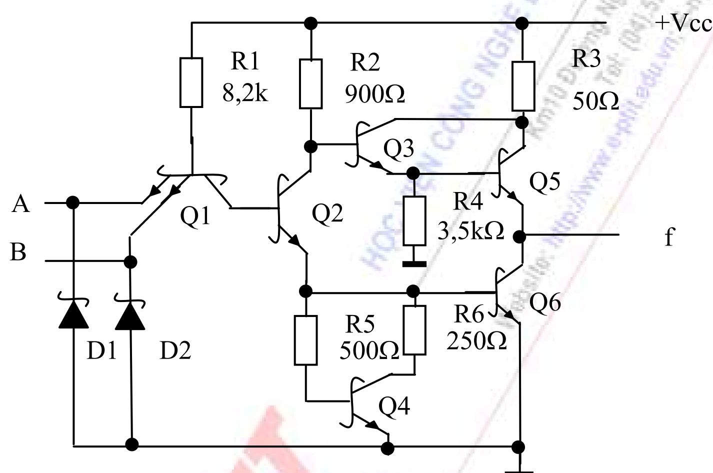
a) Kí hiệu Diode Schottky

b) Cấu tạo bán dẫn Schottky

c) Kí hiệu bán dẫn Schottky

Hình 3-10. Cấu tạo của diode Schottky

Mạch điện dùng diode Schottky chống bão hoà cho các bán dẫn như hình 3-10b. Để đơn giản, người ta gọi mạch này là bán dẫn Schottky và ký hiệu như hình 3-10c.



Hình 3-11. Mạch điện của cổng NAND 2 lối vào họ TTL+S

– **Mạch điện họ cổng TTL + S**

Nếu thay tất cả diode và bán dẫn trong mạch điện của họ TTL tiêu chuẩn bằng các diode và bán dẫn Schottky, ta sẽ có mạch điện họ cổng TTL+S. Hình 3-11 là một ví dụ về cổng NAND dùng diode Schottky.

Nhờ sử dụng diode và bán dẫn Schottky mà tần số công tác của họ cổng này tăng đáng kể. Thời gian trễ truyền lan của cổng TTL+S khoảng 3 ns, công suất tiêu thụ khoảng 19 mW.

Khi chỉ tiêu thời gian trễ không cần cao thì giá trị các điện trở phân cực được tăng lên để giảm dòng tiêu thụ của mỗi bán dẫn xuống. Họ cổng như thế có tên gọi là TTL+LS (Transistor Transistor Logic + Lowpower Schottky Diode). Công suất tiêu thụ của họ cổng này chỉ khoảng 2 mW và thời gian trễ truyền lan vẫn đạt khoảng 9,5 ns.

Nếu cần nâng cao tần số công tác, ngoài việc giảm trị số các điện trở phân cực, người ta còn dùng các cách nối mạch cải tiến. Họ cổng thu được có tên là TTL+AS.

3.1.5. Họ MOS FET

Bán dẫn trường (MOS FET) cũng được dùng rất phổ biến để xây dựng mạch điện các loại cổng logic. Đặc điểm chung và nổi bật của họ này là:

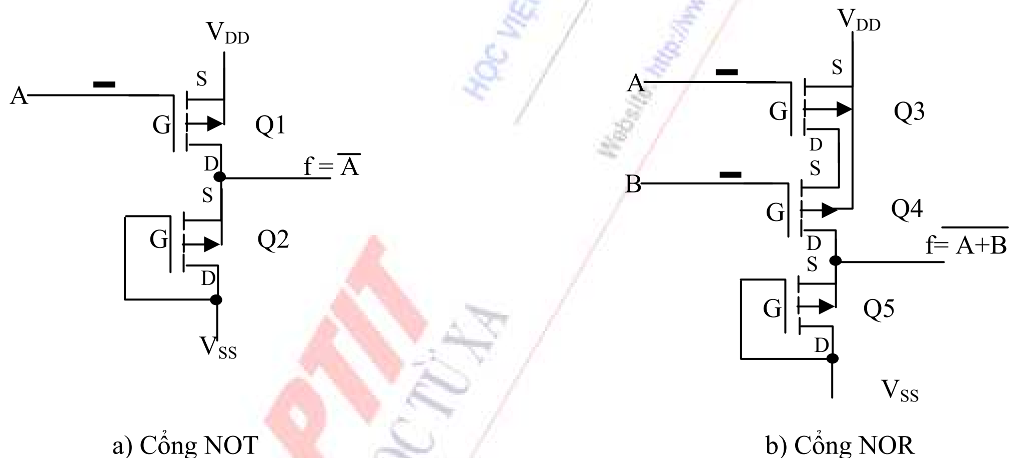
- Mạch điện chỉ bao gồm các MOS FET mà không có điện trở
- Dải điện thế công tác rộng, có thể từ +3 đến +15 V
- Độ trễ thời gian lớn, nhưng công suất tiêu thụ rất bé

Tuỳ theo loại MOS FET được sử dụng, họ này được chia ra các tiểu họ sau.

1. Loại PMOS

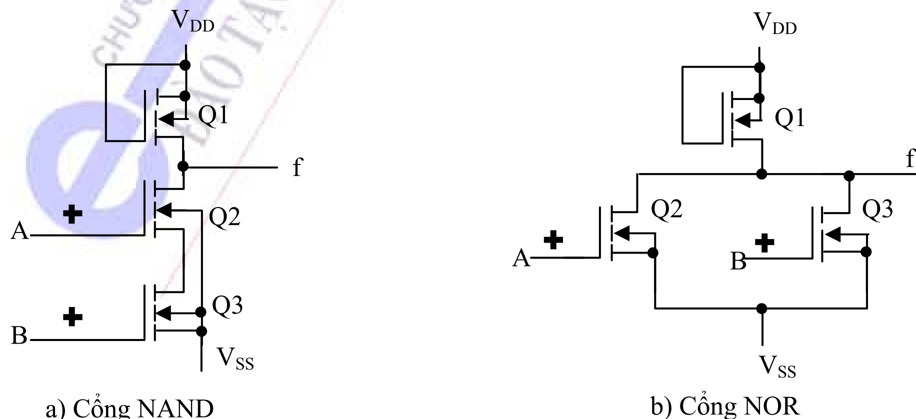
Mạch điện của họ cổng này chỉ dùng MOSFET có kênh dẫn loại P. Công nghệ PMOS cho phép sản xuất các mạch tích hợp với mật độ cao nhất.

Hình 3-12 là sơ đồ cổng NOT và cổng NOR loại PMOS. Ở đây MOSFET Q2, Q5 đóng chức năng các điện trở.



Hình 3-12. Mạch điện của cổng NOT và NOR theo công nghệ PMOS.

2. Loại NMOS



Hình 3-13. Mạch điện cổng NAND và NOR theo công nghệ NMOS.

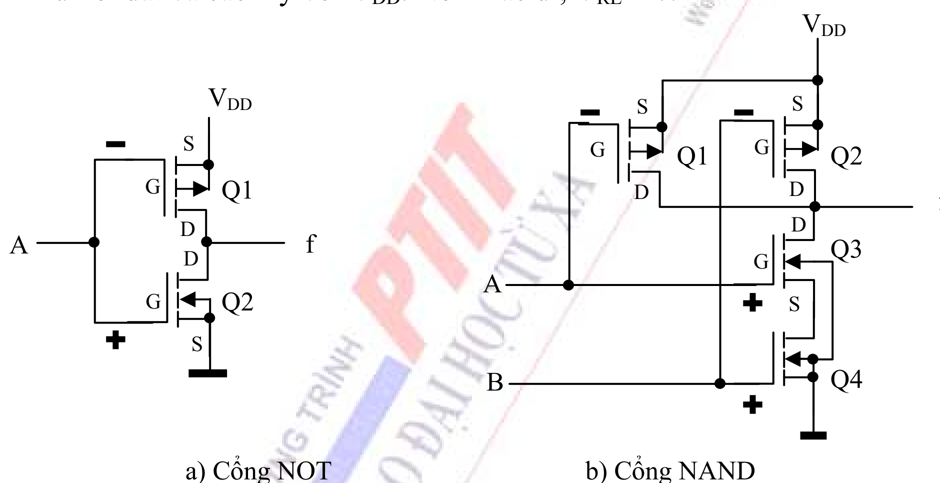
Hình 3-13 là sơ đồ cổng NAND và NOR dùng NMOS. Dấu + trên các lối vào muốn chỉ cực tính của tín hiệu kích thích. Trong trường hợp này, Q_1 cũng đóng chức năng là một điện trở. Đối với cổng NAND, ta nhận thấy rằng chỉ khi trên cả hai lối vào A và B đều lấy mức cao thì đầu ra mới có mức thấp. Ứng với 3 tổ hợp biến vào còn lại, lối ra f đều có logic thấp. Hoạt động của cổng NOR cũng được giải thích tương tự.

3. Cổng CMOS

CMOS là viết tắt các từ tiếng Anh "Complementary MOS". Mạch điện của họ cổng logic này sử dụng cả hai loại MOS FET kênh dẫn P và kênh dẫn N. Bởi vậy có hiện tượng bù dòng điện trong mạch. Chính vì thế mà công suất tiêu thụ của họ cổng, đặc biệt trong trạng thái tĩnh là rất bé.

Hình 3-14 là mạch điện của cổng NOT và NAND thuộc họ CMOS. Điểm nổi bật trong mạch điện của họ cổng này là không tồn tại vai trò các điện trở. Chức năng logic được thực hiện bằng cách thay đổi trạng thái các chuyển mạch có cực tính ngược nhau. Dấu trừ và dấu cộng trên cực cửa các MOSFET chỉ ra cực tính điều khiển chuyển mạch. Nhờ đặc điểm cấu trúc mạch, mức V_{RL}, V_{RH} đạt được gần như lý tưởng.

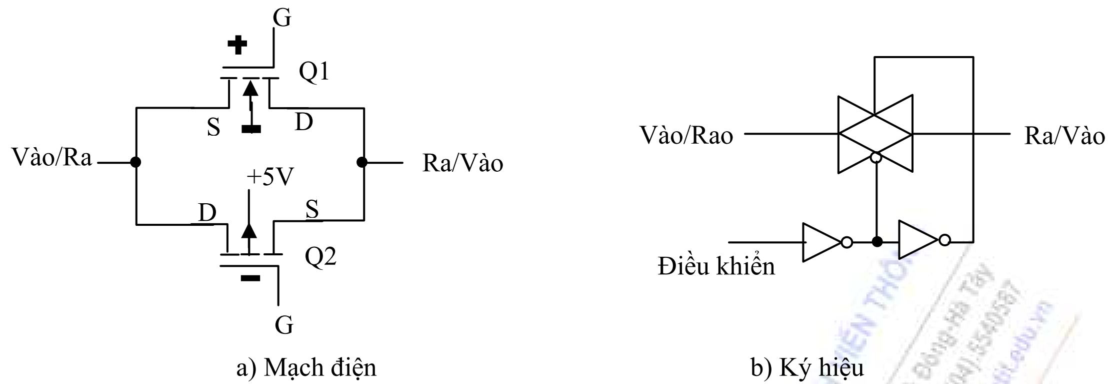
Để minh hoạ, ta có thể tìm hiểu hoạt động của cổng NOT. Từ hình 3-14a, dễ thấy rằng, nếu tác động tới lối vào A logic thấp thì Q_1 sẽ thông, Q_2 khoá. Lối ra f gần như được nối tắt tới V_{DD} và cách ly hẳn với đất, nghĩa là $V_{RH} \approx V_{DD}$. Ngược lại, khi A lấy mức cao, Q_1 mở và Q_2 đóng. Do đó, lối ra f gần như nối đất và cách ly với V_{DD}. Nói khác đi, $V_{RL} \approx 0$.



Hình 3-14. Mạch điện của họ cổng CMOS.

4. Cổng truyền dẫn

Dựa trên công nghệ CMOS, người ta sản xuất loại cổng có thể cho qua cả tín hiệu số lẫn tín hiệu tương tự. Bởi vậy cổng được gọi là cổng truyền dẫn. Sơ đồ nguyên lý và ký hiệu cổng truyền dẫn như hình 3-15.



a) Mạch điện

b) Ký hiệu

Hình 3-15. Cổng truyền dẫn.

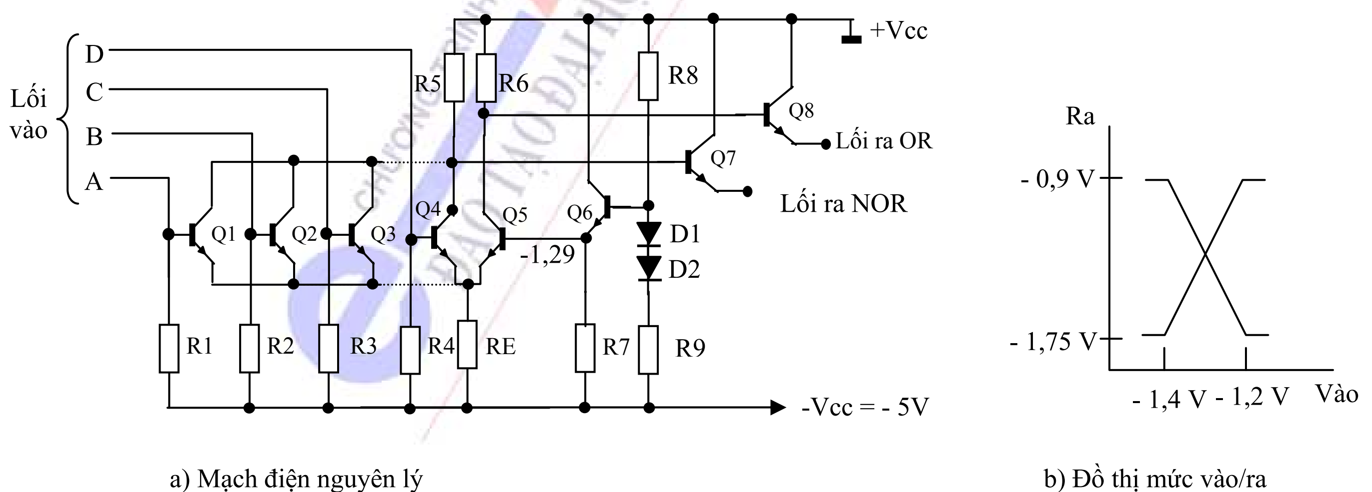
Mạch nguyên lý của cổng truyền dẫn cũng sử dụng hai MOSFET có kênh dẫn ngược nhau. Tuy nhiên cách điều khiển trạng thái các chuyển mạch lại khác với cổng logic thông thường. Trong trường hợp này, người ta phân cực sao cho khi có tín hiệu điều khiển thì cả hai chuyển mạch Q_1 và Q_2 cùng dẫn điện. Khi đó, mạch tương đương như một dây dẫn. Các cổng đảo (trong sơ đồ ký hiệu) đảm bảo cực tính điều khiển phù hợp cho cả hai cực G của mỗi MOSFET.

Tính dẫn điện của cổng truyền dẫn phụ thuộc mạnh vào tần số công tác và giá trị tải. Vì sử dụng công nghệ CMOS nên tần số công tác của cổng chỉ giới hạn ở 6 MHz.

Họ CMOS cũng có cổng D để hở và cổng ba trạng thái như họ TTL.

3.1.6- Họ ECL

ECL (Emitter Coupled Logic) là họ cổng logic có cực E của một số bán dẫn nối chung với nhau. Họ mạch này cũng sử dụng công nghệ TTL, nhưng cấu trúc mạch có những điểm khác hẳn với họ TTL. Ngoài việc sử dụng hồi tiếp âm trên điện trở R_E để chống bão hoà, mạch điện của họ ECL còn tận dụng được ưu điểm của mạch khuếch đại vi sai, nên tần số công tác họ này là cao nhất trong các họ. Ngoại trừ thời gian trễ, tất cả các tham số còn lại đều kém hơn các họ khác.



a) Mạch điện nguyên lý

b) Đồ thị mức vào/ra

Hình 3-16. Cổng OR/NOR thuộc họ ECL.

Hình 3-16 là mạch điện và đồ thị mức vào ra của một cổng OR/NOR thuộc họ ECL. Vì điện thế ở trên hai cực collector của Q_4, Q_5 là bù nhau nên có thể lấy ra ở cực E của Q_7 chức năng OR và ở cực E của Q_8 chức năng NOR. Để mạch hoạt động theo logic mức âm, +Vcc được nối đất, -

Vcc được nối tới âm nguồn. Mức logic trong mạch được biến đổi từ giá trị thấp là -1,75 V đến giá trị cao là - 0,9 V so với điện thế đất. Khi muốn có mức logic ra dương các cực E nối tới đất.

3.2. GIAO TIẾP GIỮA CÁC CỔNG LOGIC CƠ BẢN TTL-CMOS VÀ CMOS-TTL

Trong nhiều ứng dụng, yêu cầu chuyển đổi các tín hiệu giữa các mức logic khác nhau như từ TTL sang CMOS hoặc ngược lại. Các cổng logic collector hở hoặc các mạch khuếch đại transistor đơn giản thường được sử dụng trong các mạch chuyển đổi này.

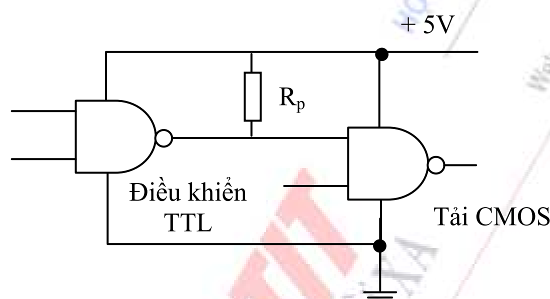
3.2.1. Giao tiếp giữa TTL và CMOS.

Để tạo được giao tiếp giữa TTL và CMOS thì ta phải để ý đến nguồn cung cấp của 2 họ. Họ TTL cần điện áp cung cấp là + 5V, họ CMOS có thể dùng điện áp cung cấp từ +3V đến +15V.

a. Cùng điện áp cung cấp +5V.

Trong trường hợp này điện áp ra của TTL nhỏ hơn so với điện áp vào của CMOS. Do vậy ta phải dùng mạch bổ sung để tương hợp hai loại IC khác nhau.

Giải pháp tiêu chuẩn là dùng điện trở kéo lên giữa điều khiển TTL và tải CMOS như hình 3-17.



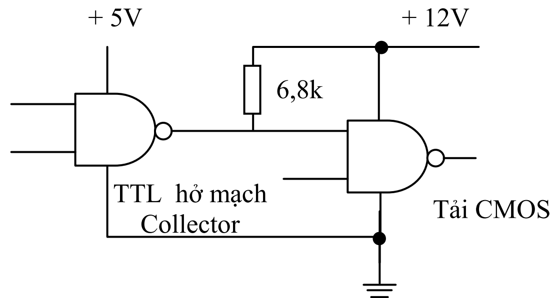
Hình 3-17. Điều khiển TTL và tải CMOS

b. Khác điện áp cung cấp.

Điện áp cung cấp dùng cho IC CMOS thích hợp nhất là từ +9V đến +12V. Một cách dùng để điện áp cung cấp lớn là sử dụng IC TTL hở mạch Collector như ở hình 3-18, vì tầng ra của TTL hở cực C chỉ gồm transistor nhận dòng với cực C thả nổi. Ở hình này cực C để hở được nối với nguồn cung cấp +12V qua điện trở kéo lên 6,8kΩ. Khi lối ra của họ TTL ở mức L thì dòng của nó là:

$$I_{\text{nhận dòng}} = \frac{12V}{6,8k\Omega} = 1,76mA$$

Khi lối ra của TTL ở mức H thì lối ra của cực C để hở tăng lên một cách thụ động đến +12V. Trong trường nào thì các lối ra của TTL cũng đều tương hợp với các trạng thái ở lối vào của CMOS.

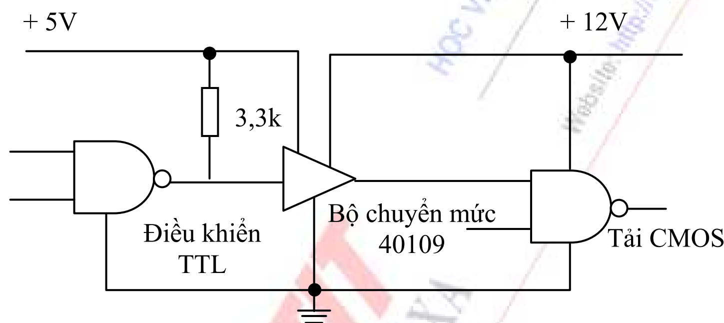


Hình 3-18. Điều khiển TTL hở mạch Collector và tải CMOS

c. Bộ chuyển mức nguồn dùng CMOS.

Hình 3-19 là bộ chuyển mức CMOS 40109. Tầng lối vào của IC dùng điện áp cung cấp +5V trong khi tầng lối ra dùng +12V.

Trong hình 3-19, IC TTL tiêu chuẩn điều khiển bộ chuyển mức nguồn, nó kéo IC TTL lên ít nhất là +2,4V. Điện trở kéo lên tiếp tục đưa điện áp lên cao đến mức +5V, mức này đảm bảo chắc chắn lối vào ở mức H. Lối ra của bộ chuyển mức nối với nguồn +12V.



Hình 3-19. Bộ chuyển mức CMOS cho phép sử dụng hai loại nguồn +5V và +12V.

3.2.2. Giao tiếp giữa CMOS và TTL

Để tạo ra được giao tiếp giữa họ CMOS và TTL thì ta phải quan tâm đến vấn đề chuyển mức điện áp cho tới khi trạng thái lối ra của CMOS phù hợp với lối vào của TTL. Ta phải đảm bảo chắc chắn lối ra ở trạng thái L của CMOS luôn luôn nhỏ hơn 0,8 V(đây là điện áp lối vào lớn nhất ở trạng thái L của họ TTL). Điện áp lối ra ở trạng thái H của CMOS luôn luôn lớn hơn 2 V(đây là điện áp lối vào nhỏ nhất ở trạng thái H của họ TTL).

a. Cùng điện áp cung cấp +5V.

Theo số liệu kỹ thuật của IC 74Cxx thì trường hợp xấu nhất dòng lối ra của CMOS điều khiển TTL là:

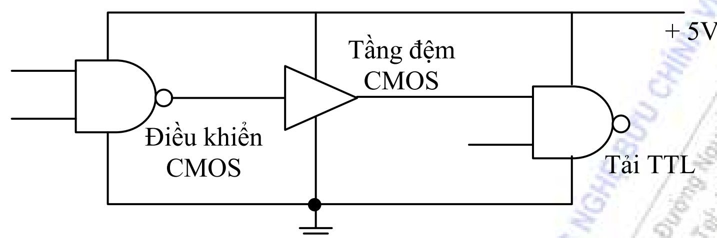
$I_{OL\,MAX} = 360\mu A$; $I_{OH\,MAX} = -360\mu A$

Điều này có nghĩa là điều khiển CMOS có thể cho nhận dòng là 360 μA khi ở trạng thái L, đó là dòng vào đối với IC TTL loại Schottky công suất thấp. Mặt khác, điều khiển CMOS có thể cho dòng nguồn 360 μA, nó lớn hơn mức cần thiết để điều khiển dòng vào ở trạng thái H. Như vậy hệ số ghép tải giữa CMOS và 74LS là bằng 1.

Đối với loại IC TTL công suất thấp thì có dòng lối vào là 180 μA thì hệ số ghép tải giữa CMOS và 74L là bằng 2.

IC CMOS không thể điều khiển trực tiếp IC TTL tiêu chuẩn, vì dòng lối vào ở trạng thái L yêu cầu là 1,6 mA, mà transistor nhận dòng của IC CMOS có điện trở xấp xỉ 1,11kΩ (trường hợp xấu nhất). Nên điện áp lối ra của IC CMOS bằng 1,6 mA x 1,11kΩ = 1,78 V. Điện áp này quá lớn đối với lối vào ở trạng thái L của IC TTL.

- Dùng tầng đệm bằng CMOS.



Hình 3-20. Tầng đệm CMOS có thể điều khiển tải TTL tiêu chuẩn

Hình 3-20 là mạch điều khiển IC CMOS với hệ số tải qua tầng đệm. Tầng đệm có dòng ra lớn. Ví dụ IC 74C902 có 6 tầng đệm CMOS, mỗi tầng đệm có dòng ở lối ra trong trường hợp xấu nhất là:

$I_{OL\ MAX}$ = 3.60mA

$I_{OH\ MAX}$ = 800μA

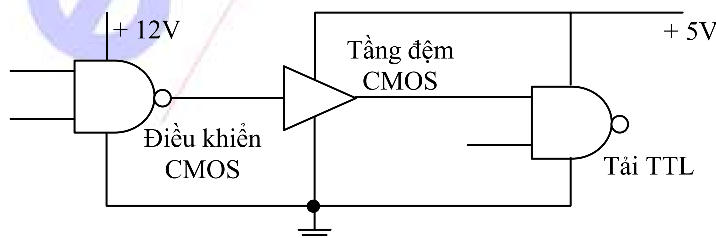
Vì tải TTL tiêu chuẩn có dòng lối vào ở trạng thái L bằng 1,6mA và dòng lối vào ở trạng thái H là 48 μA, IC 74C902 có thể điều khiển hai tải TTL tiêu chuẩn.

Các IC khác được dùng làm tầng đệm như hình 5-19 là IC CD4049A, 4050: đảo; CD405CA: không đảo, 74C901: đảo…

b. Khác điện áp cung cấp.

Các tầng đệm CMOS như 74C902 có thể dùng điện áp cung cấp từ +3V đến +15V và điện áp lối vào từ -0,3 V đến +15V> Điện áp lối vào có thể lớn hơn điện áp cung cấp mà không làm hỏng loại IC dùng làm tầng đệm này. Ví dụ ta có thể dùng điện áp lối vào ở trạng thái H là +12V ngay khi điện áp cung cấp chỉ bằng 5V.

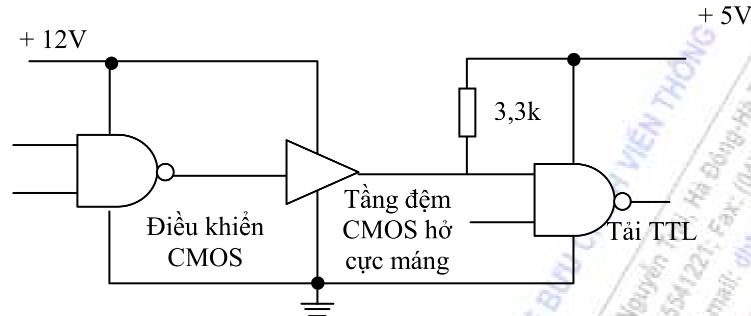
Hình 5-23 là mạch điều khiển CMOS dùng điện áp cung cấp +12V, trong khi tầng đệm CMOS có điện áp cung cấp là +5V.



Hình 3-21. Điều khiển CMOS hoạt động thích hợp nhất với nguồn cung cấp +12V.

c. Giao diện của hở cực máng.

Ta đã biết ở IC TTL hở mạch Collector, tầng lối ra của transistor nhận dòng với cực C thả nổi. Tương tự như vậy đối với IC CMOS cũng có hở cực máng. Ví dụ: IC 74C906 có 6 tầng đệm hở cực máng.



Hình 3-22. Tầng đệm CMOS hở cực máng làm tăng dòng nhận.

Hình 3-22 là mạch dùng tầng đệm CMOS hở cực máng làm giao diện điều khiển CMOS và tải TTL. Điện áp cung cấp cho hầu hết các tầng đệm là +12V. Tuy vậy có thể nối tầng đệm hở cực máng với nguồn cung cấp +5V qua một điện trở kéo lên (pull up) có giá trị 3,3kΩ. Cách nối này có ưu điểm là cả điều khiển CMOS và tầng đệm CMOS đều được cung cấp nguồn +12V, không kể lối ra hở cực máng giao diện với TTL

TÓM TẮT

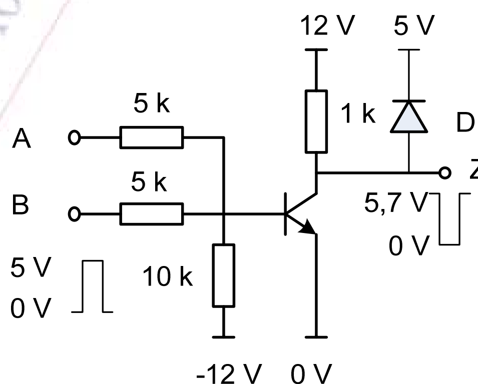
Chương 3 đã trình bày cấu trúc, nguyên lý và đặc điểm của cổng thường dùng. Xuất phát từ thực tế mạch điện đã vi mạch hoá, nên trọng tâm chú ý nghiên cứu của chúng ta là các cổng được vi mạch hoá.

Có 2 loại vi mạch số phổ biến nhất : TTL và MOS. TTL là công nghệ điển hình trong nhóm công nghệ transistor bao gồm TTL, HTL, ECL…, MOS là công nghệ vi mạch sử dụng MOSFET, trong đó điển hình là MOS…

Đồng thời trong chương 3 cũng đưa ra vấn đề giao tiếp giữa các họ cổng đó với nhau.

CÂU HỎI ÔN TẬP

1. Chức năng của mạch logic RTL có sơ đồ như hình vẽ sau:



a. NOR

b. OR

c. AND

d. NAND

2. Với mạch có sơ đồ như trong câu hỏi 1, nhưng điện áp logic lối vào tương ứng với các mức logic cao và thấp lần lượt là 10 V và 0 V thì chức năng của mạch là gì?

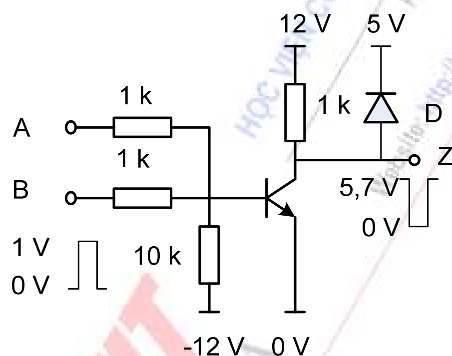
a. NOR

b. OR

c. AND

d. NAND

3. Cho mạch có sơ đồ như sơ đồ sau, điện áp logic lối và tương ứng với các mức logic cao và thấp lần lượt là 1 V và 0 V, nêu chức năng của mạch?



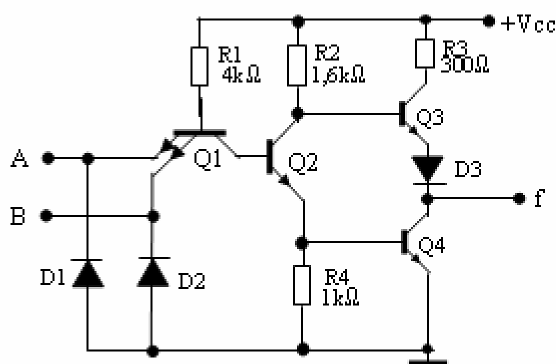
a. NOR

b. OR

c. AND

d. NAND

4. Chức năng của diode D_3 trong sơ đồ sau là gì?



a. Cách ly transistor Q_3 và Q_4

b. Dịch mức điện áp làm cho Q_3 và Q_4 không bao giờ cùng đóng hoặc cùng mở

c. Chống nhiễu lối ra

d. Cách ly Q_4 khỏi mạch ngoài nối vào đầu ra f

5. Chức năng của mạch biểu diễn trong sơ đồ như câu hỏi 4 sẽ thay đổi thế nào nếu diode D_3 chuyển tới chân base của transistor Q_3 (cathode D_3 nối với base Q_3 còn anode nối với collector Q_2)?

   a. Q_3 luôn cấm

   b. Q_3 luôn mở

   c. Chức năng của mạch không thay đổi

   d. Lối ra luôn ở trạng thái treo

6. Cổng collector hở sẽ hoạt động bình thường như các cổng logic bình thường nếu :

   a. Lối ra được nối lên nguồn thông qua một trở gánh

   b. Lối ra được nối lên nguồn thông qua một tụ gánh

   c. Lối ra nối xuống đất thông qua một trở

   d. Lối ra nối xuống đất thông qua một tụ

7. Tác dụng của trạng thái trở kháng lối ra cao trong cổng ba trạng thái là :

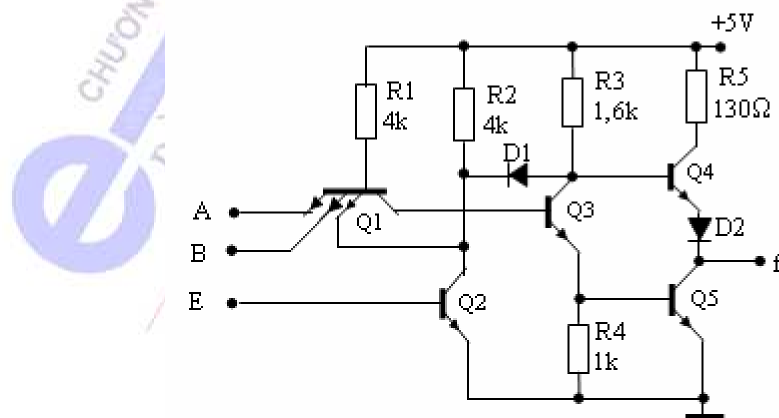
   a. Đưa ra mức logic thứ 3 là trung bình của hai mức cao và thấp

   b. Cách ly giữa các lối ra của các cổng logic khi chúng cùng được nối vào một lối vào

   c. Có mức logic thấp nhưng trở kháng cao

   d. Có mức logic cao nhưng trở kháng cao

8. Mạch điện được biểu diễn trong sơ đồ sau có còn hoạt động như bình thường không nếu như diode D_1 bị nối tắt ?



   a. Mạch trở thành cổng NAND với hai trạng thái lối ra như các cổng NAND thường

   b. Mạch trở thành cổng NOR

c. Trạng thái lối ra không theo logic cơ bản nào

d. Vẫn hoạt động bình thường là cổng NAND 3 trạng thái

9. Mạch điện như trong câu hỏi 8 có còn hoạt động như bình thường không nếu như điện trở R_4 có giá trị bằng 10 k?

   a. Nó sẽ hoạt động như mạch NOR

   b. Nó sẽ hoạt động như mạch XOR

   c. Vẫn hoạt động bình thường

   d. Cả ba cách trả lời trên đều sai

10. Với mạch điện TTL như sơ đồ trong câu hỏi 4, hiện tượng gì sẽ xảy ra khi một trong hai lối vào để lửng?

   a. Lối vào này được tính logic 0

   b. Lối vào này được tính logic 1

   c. Mạch không hoạt động

   d. Cả ba cách trả lời trên đều sai

11. So sánh cổng NOT họ MOS và CMOS ta thấy :

   a. Công suất tiêu thụ của MOS cao hơn CMOS

   b. Công suất tiêu thụ của CMOS cao hơn MOS

   c. Công suất tiêu thụ của hai họ như nhau

   d. Cả ba cách trả lời trên đều sai

12. Có cho phép đầu vào của mạch CMOS để lơ lửng không? Có thể nói đầu vào để lửng tương đương với mức cao không?

   a. Được- Có thể coi là mức 1

   b. Được- Phải coi là mức 0

   c. Không được- Để mạch hoạt động bình thường thì đầu vào không dùng phải nối với mức logic 0

   d. Không được- Để mạch hoạt động bình thường thì đầu vào không dùng phải nối với mức logic 1

13. Cổng truyền dẫn là cổng

   a. Chỉ cho phép tín hiệu số đi qua theo một chiều nhất định

   b. Chỉ cho phép tín hiệu số đi qua theo hai chiều

   c. Chỉ cho phép tín hiệu tương tự đi qua theo một chiều nhất định

   d. Cho phép tín hiệu tương tự đi qua theo hai chiều

14. Ưu điểm của các cổng logic họ ECL là

   a. Tần số công tác nhanh

   b. Điện áp nguồn nuôi thấp

   c. Công suất tiêu thụ thấp

   d. Độ chống nhiễu cao



CHƯƠNG 4: MẠCH LOGIC TỔ HỢP

GIỚI THIỆU CHUNG

Các hàm logic được thực hiện nhờ các hệ vật lý gọi là các *hệ logic* hay là các *mạch logic*. Trong chương 4 chúng ta đề cập đến các mạch logic tổ hợp, tức là các mạch mà tín hiệu ở đầu ra chỉ phụ thuộc vào tín hiệu ở đầu vào của mạch tại thời điểm đang xét. Nói cách khác, các tín hiệu ra không phụ thuộc vào "lịch sử " của tín hiệu vào trước đó, nghĩa là các hệ này làm việc theo nguyên tắc không có nhớ. Hoạt động của các mạch tổ hợp được mô tả bằng các bảng trạng thái hoặc bằng các hàm chuyển mạch logic đặc trưng cho quan hệ giữa các đại lượng vào và ra của hệ thống. Về mặt cấu trúc, các mạch tổ hợp không chứa một thiết bị hoặc một phần tử nhớ thông tin nào cả.

Trong chương này đề cập đến các mạch điện cụ thể thực hiện các chức năng khác nhau của hệ thống số. Các mạch điện này được thiết kế dựa trên các cổng logic tổ hợp. Các cổng logic này được tích hợp trong một IC cỡ vừa (MSI) có chứa khoảng vài chục tới vài trăm các các cổng logic cơ sở đó được xét đến ở chương 4. Những linh kiện này được chế tạo nhằm thực hiện một số các hoạt động thu nhận, truyền tải, biến đổi các dữ liệu thông qua tín hiệu nhị phân, xử lý chúng theo một phương thức nào đó.

Phần đầu của chương giới thiệu cách phân tích và thiết kế các mạch logic tổ hợp đơn giản.

Phần tiếp theo giới thiệu về Hazard trong mạch logic tổ hợp. Đây là phần rất quan trọng khi thiết kế mạch. Nếu không để ý đến hiện tượng này có thể dẫn đến sự làm việc sai lệch của cả hệ thống. Phân tích và nhận dạng Hazard có ý nghĩa rất quan trọng không những trong tổng hợp các hệ logic mà cả trong tự động chẩn đoán trạng thái làm việc của chúng.

Phần tiếp theo giới thiệu một số mạch tổ hợp thông dụng trong các hệ thống số:

- Mã hoá và giải mã các luồng dữ liệu nhị phân.

- Hợp kênh và phân kênh để chọn hoặc chia tách các luồng số nhị phân theo những yêu cầu nhất định để định tuyến cho chúng trong việc truyền dẫn thông tin,

- Các mạch cộng, trừ.

- Các phép so sánh số để đánh giá định tính và định lượng trọng số của các số nhị phân.

- Mạch tạo và kiểm tra tính chẵn lẻ.

- Đơn vị số học và logic (ALU).

NỘI DUNG

4.1 KHÁI NIỆM CHUNG

Căn cứ vào đặc điểm và chức năng logic, các mạch số được chia thành 2 loại chính: mạch tổ hợp và mạch tuần tự (mạch tuần tự được trình bày ở chương sau).

1) Đặc điểm cơ bản của mạch tổ hợp

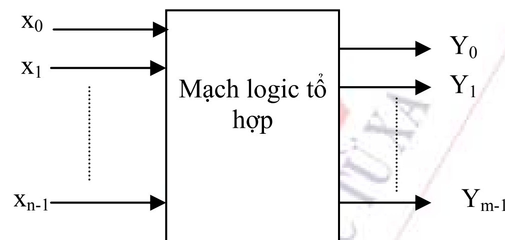
Trong mạch số, mạch tổ hợp là mạch mà trị số ổn định của tín hiệu đầu ra ở thời điểm đang xét chỉ phụ thuộc vào tổ hợp các giá trị tín hiệu đầu vào. Đặc điểm cấu trúc mạch tổ hợp là được cấu trúc nên từ các cổng logic. Vậy các mạch điện cổng ở chương 2 và các mạch logic ở chương 3 đều là các mạch tổ hợp.

2) Phương pháp biểu diễn chức năng logic

Các phương pháp thường dùng để biểu diễn chức năng logic của mạch tổ hợp là hàm số logic, bảng trạng thái, sử dụng logic, bảng Cac nô (Karnaugh), cũng có khi biểu thị bằng đồ thị thời gian dạng xung.

Đối với vi mạch cỡ nhỏ (SSI) thường biểu diễn bằng hàm logic. Đối với vi mạch cỡ vừa (MSI) thường biểu diễn bằng bảng trạng thái.

Sơ đồ khối tổng quát của mạch logic tổ hợp được trình bày ở hình 4-1.



Hình 4-1 Sơ đồ khối tổng quát của mạch logic tổ hợp.

Như vậy, mạch logic tổ hợp có thể có *n* lối vào và *m* lối ra. Mỗi lối ra là một hàm của các biến vào. Quan hệ vào, ra này được thể hiện bằng hệ phương trình tổng quát sau:

$$Y_0 = f_1(x_0, x_1, ..., x_{n-1});$$

$$Y_1 = f_2(x_0, x_1, ..., x_{n-1});$$

$$\dots$$

$$Y_{m-1} = f_{m-1}(x_0, x_1, ..., x_{n-1}).$$

Từ đó, ta thấy rằng đặc điểm nổi bật của mạch logic tổ hợp là hàm ra chỉ phụ thuộc các biến vào mà không phụ thuộc vào trạng thái của mạch. Cũng chính vì thế, trạng thái ra chỉ tồn tại trong thời gian có tác động vào.

Thể loại của mạch logic tổ hợp rất phong phú. Phạm vi ứng dụng của chúng cũng rất rộng.

4.2 PHÂN TÍCH MẠCH LOGIC TỔ HỢP

Phân tích mạch logic tổ hợp là đánh giá, phê phán một mạch đó. Trên cơ sở đó, có thể rút gọn, chuyển đổi dạng thực hiện của mạch điện để có được lời giải tối ưu theo một nghĩa nào đấy.

Mạch tổ hợp có thể bao gồm hai hay nhiều tầng, mức độ phức tạp của của mạch cũng rất khác nhau.

Nếu mạch đơn giản thì ta tiến hành lập bảng trạng thái, viết biểu thức, rút gọn, tối ưu (nếu cần) và cuối cùng vẽ lại mạch điện.

Nếu mạch phức tạp thì ta tiến hành phân đoạn mạch để viết biểu thức, sau đó rút gọn, tối ưu (nếu cần) và cuối cùng vẽ lại mạch điện.

4.3 THIẾT KẾ MẠCH LOGIC TỔ HỢP

Thiết kế là bài toán ngược với bài toán phân tích. Nội dung thiết kế được thể hiện theo tuần tự sau:

1- Phân tích bài toán đã cho để gắn hàm và biến, xác lập mối quan hệ logic giữa hàm và các biến đó;

2- Lập bảng trạng thái tương ứng;

4- Từ bảng trạng thái có thể viết trực tiếp biểu thức đầu ra hoặc thiết lập bảng Cac nô tương ứng;

4- Dùng phương pháp thích hợp để rút gọn, đưa hàm về dạng tối giản hoặc tối ưu theo mong muốn;

5- Vẽ mạch điện thể hiện.

Ví dụ : Một ngôi nhà hai tầng. Người ta lắp hai chuyển mạch hai chiều tại hai tầng, sao cho ở tầng nào cũng có thể bật hoặc tắt đèn. Hãy thiết kế một mạch logic mô phỏng hệ thống đó?

Lời giải:

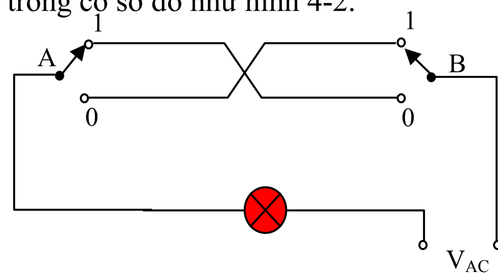
+ Nếu ký hiệu hai công tắc là hai biến A, B. Khi ở tầng 1 ta bật đèn và lên tầng 2 thì tắt đèn đi và ngược lại. Như vậy đèn chỉ có thể sáng ứng với hai tổ hợp chuyển mạch ở vị trí ngược nhau. Còn đèn tắt khi ở vị trí giống nhau. Hệ thống chiếu sáng trong có sơ đồ như hình 4-2.

Bảng trạng thái mô tả hoạt động của hệ như chỉ ở bảng 4-1.

Biểu thức của hàm là: $f = \overline{A}B + A\overline{B} = A \oplus B$

hoặc

$f = \overline{\overline{\overline{AB}\,A}\;\;\overline{\overline{AB}\,B}}$

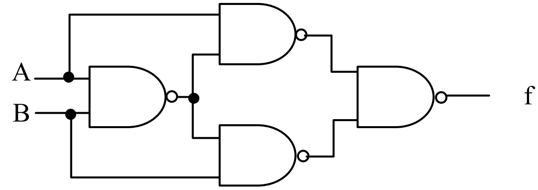


Hình 4-2 Mạch điện của hệ thống chiếu sáng

Đây là hàm cộng XOR đã quen thuộc ở các chương trước. Hàm này có thể được thể hiện bằng nhiều kiểu mạch khác nhau. Hình 4-3 là một dạng sơ đồ thể hiện hàm f.

A	B	f
0	0	0
0	1	1
1	0	1
1	1	0

Bảng 4-1. Bảng trạng thái mô tả hoạt động của hệ chiếu sáng



Hình 4-3. Sơ đồ logic thể hiện hàm f

4.4 HAZARD TRONG MẠCH TỔ HỢP

4.4.1. Khái niệm.

Việc thiết kế các mạch logic nhìn chung không phức tạp, vì cần có biểu thức toán là ta có thể vẽ ra được mạch điện và lắp ráp thành hệ thống điều khiển. Trên thực tế, không phải mạch nào cũng có thể hoạt động tốt được, nguyên nhân là do cấu trúc của mạch tổ hợp gây ra, hiện tượng hoạt động không ổn định xảy ra trong mạch tổ hợp được gọi là *hazard.*

Hazard còn được gọi là sự "sai nhầm", hoạt động lúc được lúc không của mạch logic. Sự "sai nhầm" này có thể xảy ra trong một mạch điện hoàn toàn không có hỏng hóc linh kiện. Tức là trong mạch, các linh kiện hoàn toàn tốt nhưng điều khiển chức năng lúc được lúc không. Nói chung là mạch hoạt động không có sự tin cậy. Hiện tượng của Hazard trong mạch tổ hợp có thể gặp là:

- Hazard chỉ xuất hiện một lần và không bao giờ gặp lại nữa.

- Hazard có thể xuất hiện nhiều lần (theo một chu kỳ nào đó hoặc không theo một chu kỳ nào).

- Hazard có thể do chính chức năng của mạch điện gây ra. Đây là trường hợp khó giải quyết nhất khi thiết kế.

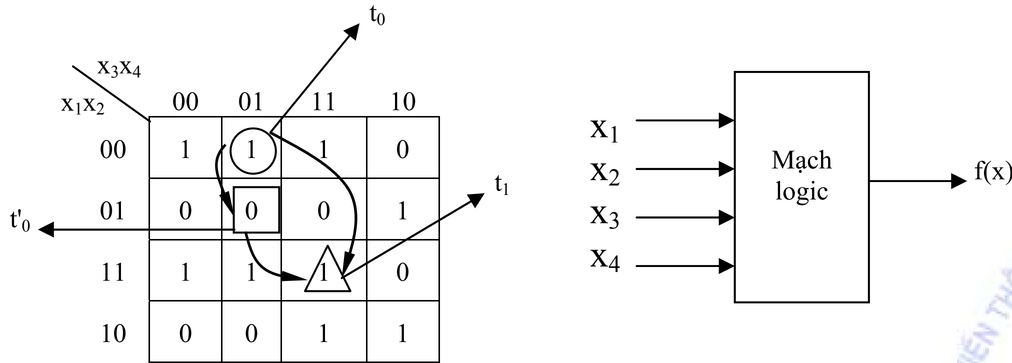
Như ta đã biết, một trong các đặc tính quan trọng nhất của mạch điện khi hoạt động là quán tính, độ linh động hay sự chậm trễ của mạch. Chính sự chậm trễ này làm cho tín hiệu từ đầu vào không thể truyền ngay tức khắc tới đầu ra của mạch điện, điều này làm cho các thiết bị điều khiển phía sau không thể có phản ứng tức khắc đối với tín hiệu đưa vào. Do tất cả các mạch điện đều có thời gian trễ nhất định, ngay cả ở các mạch vi điện tử cũng có thời gian trễ. Sự thay đổi nhiệt độ môi trường cũng làm cho thời gian trễ thay đổi, dẫn đến sự sai lệch khi điều khiển của mạch logic, đó chính là hazard.

4.4.2. Bản chất của Hazard

Để hiểu được nguyên nhân xuất hiện hazard trong mạch logic tổ hợp, hazard chỉ xuất hiện trong mạch tổ hợp mà không xuất hiện ở bất kỳ hệ thống điện tử nào khác. Ta xét ví dụ sau:

Giả sử tín hiệu vào là $X = (x_1, x_2, x_3, x_4)$ thay đổi giá trị từ (0 0 0 1) đến (1 1 1 1), tức là (X) thay đổi từ $Q \rightarrow P$. Nhìn vào bảng Cac nô (hình 4-4) ta thấy đáp ứng ra của mạch logic tổ hợp khi tín hiệu vào bị thay đổi có giá trị:

$$f(Q) = f(0001) = \textcircled{1} \rightarrow f(P) = f(1111) = \triangle 1$$



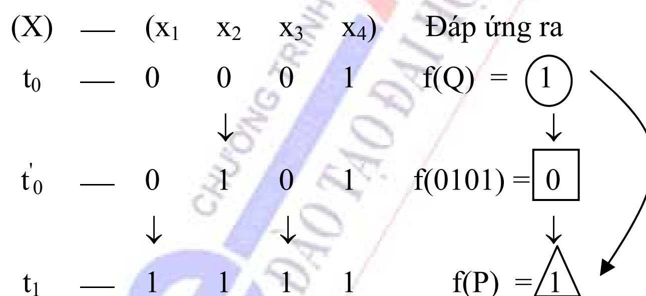
Hình 4-4. Mạch chức năng logic

Như vậy tín hiệu vào (X) thay đổi giá trị từ Q = (0001) đến P = (1111) làm cho đáp ứng ra của mạch bị thay đổi giá trị từ (1) sang △1 , sự thay đổi điều khiển ở đầu ra của mạch theo sự thay đổi tín hiệu vào (X) → điều này hoàn toàn chính xác, khi đó hazard không xuất hiện và không xảy ra điều khiển bị sai nhầm.

Nhưng thực tế có thể không được như vậy vì khi tín hiệu vào thay đổi từ Q = (0001) đến P = (1111), ta thấy tín hiệu x_1, x_2, x_3 bị thay đổi còn giá trị x_4 không bị thay đổi. Mạch điện nào cũng xuất hiện thời gian trễ là (τ) và sự thay đổi giá trị (0→1 hay 1→0) của tín hiệu đều có thời gian trễ nhất định.

Trong trường hợp này, các tín hiệu vào (x_1, x_2, x_3) có giá trị logic bị thay đổi khi ta thay đổi bộ tín hiệu vào, và chúng sẽ có một thời gian trễ nhất định (có thể rất nhỏ, cỡ μs hay ns). Mặt khác, thời gian trễ của mỗi đường tín hiệu vào (x_i) lại khác nhau, dù cùng một chủng loại IC. Như vậy nếu (x_1, x_2, x_3) được thay đổi đồng thời và chúng có thời gian trễ khác nhau thì vẫn xảy ra hiện tượng "chạy đua" của tín hiệu vào tới đầu ra của mạch điện.

Vì có sự "chạy đua" giữa ba tín hiệu vào (x_1, x_2, x_3) (x_4 không thay đổi nên không đua), giả sử x_2 chạy nhanh hơn (có thời gian trễ nhỏ hơn) x_1, x_2 (giả sử thời gian trễ của hai tín hiệu này bằng nhau). Mối quan hệ này ta có thể biểu diễn như sau:



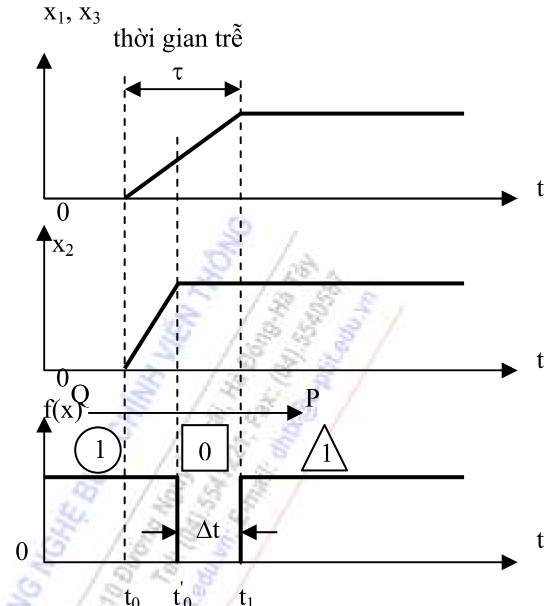
Do x_2 "chạy" nhanh hơn x_1 và x_3 nên giá trị của x_2 chuyển từ 0 sang 1 trước giá trị của x_1 và x_3. Sau một thời gian thì (x_1, x_3) mới chuyển từ 0 sang 1.

Quan hệ "chạy đua" giữa ba tín hiệu vào được minh hoạ bằng biểu đồ sau:

Do x_2 "chạy nhanh" hơn (x_1, x_3) nên trong khoảng thời gian Δt đã xuất hiện một xung zêrô nhất thời. Như vậy trong thời gian trễ τ của mạch tín hiệu ra đã thay đổi từ 1→0→1 (đúng ra là không được thay đổi), tạo ra một xung kim nhất thời. Hiện tượng xuất hiện một xung zêrô ở đầu ra của mạch được gọi là hiện tượng hazard và đây là hazard nhất thời, nó chỉ xuất hiện trong thời gian trễ τ sau đó lại mất ngay. Như vậy ta có thể nói rằng sự "chạy đua" của tín hiệu vào gây ra

hazard, hay thời gian trễ của mạch sẽ làm xuất hiện hazard, đó là tín hiệu điều khiển không mong muốn ở đầu ra.

Xung Hazard là một xung kim xuất hiện ở đầu ra của mạch logic tổ hợp, vì thời gian xuất hiện (Δt) nhỏ hơn thời gian trễ của mạch (τ) nên xung hazard có thể xuất hiện nhưng không gây nguy hiểm, không gây ra sự điều khiển sai nhầm. Vì xung hazard quá hẹp nên năng lượng của nó không đủ lớn để có thể kích nhầm hay kích được các mạch điện tiếp theo, do đó dù có xung hazard nhưng mạch điện vẫn hoạt động tốt. Xung hazard chỉ thật sự nguy hiểm khi độ rộng Δt đủ lớn thì nó có đủ năng lượng để lật chuyển mạch điện tiếp theo gây ra hiện tượng điều khiển nhầm.



Hình 4-5. Hiện tượng hazard

Như vậy có thể thấy với bộ tín hiệu vào thay đổi kiểu khác với tổ hợp trên thì có thể không xuất hiện xung hazard. Hay với một chức năng khác dù có hiện tượng "chạy đua" tín hiệu vào giữa (x_1,x_3 và x_2) như ví dụ trên nhưng có f(0101) = 1 thì hazard cũng không thể xuất hiện do xung zêrô nhất thời không có.Do vậy ta thấy hiện tượng hazard xuất hiện rất ngẫu nhiên cho dù mạch điện chứa toàn các linh kiện tốt.

4.4.3. Phân loại.

Đầu tiên ta đề cập đến một số định nghĩa tên gọi khi nói về hazard như sau:

$Q = (q_1, q_2, ...q_k, q_{k+1},...q_n)$

$P = (\overline{q_1}, \overline{q_2} ...\overline{q_k}, q_{k+1},...q_n)$

Ở đây P và Q là tập tín hiệu vào của mạch, nhưng yêu cầu giữa P và Q cần có số lượng vị trí thay đổi giá trị logic ≥ 2, vì chỉ khi tập tín hiệu vào thay đổi giá trị logic đồng thời với ít nhất 2 vị trí (2 biến số) thì mới xuất hiện hiện tượng "chạy đua" tín hiệu vào, và khi đó hazard mới có khả năng xuất hiện. Còn nếu tín hiệu vào chỉ thay đổi giá trị lần lượt trên từng đầu vào một thì sẽ không có hiện tượng chạy đua tín hiệu và hazard không thể xuất hiện được.

Định nghĩa 1: Nếu tập tín hiệu vào (X) thay đổi từ Q sang P thì được gọi là có sự chuyển đổi từ Q sang P (Q → P).

Định nghĩa 2: Hazard nhất thời xuất hiện trong mạch logic tổ hợp là hiện tượng tín hiệu ra ở một hoặc nhiều đầu ra của mạch xuất hiện khác với các giá trị quy định cho chúng theo hàm Boole trong thời gian chuyển đổi từ Q → P.

Định nghĩa 3: Hazard nhất thời xuất hiện trong mạch logic tổ hợp trong thời gian chuyển đổi từ Q → P gọi là *hazard tĩnh* nếu và chỉ nếu *f(Q) = f(P)*. Ở đây *f(X)* là hàm logic được thực hiện bởi các mạch đã cho.

Định nghĩa 4: Hazard nhất thời xuất hiện trong mạch logic tổ hợp trong thời gian chuyển đổi từ Q → P gọi là *hazard động* nếu và chỉ nếu $f(Q) = \overline{f(P)}$. Như vậy khi có hazard nhất thời

động thì tín hiệu ở đầu ra thay đổi ít nhất ba lần, ví dụ 1→0→1→0, nghĩa là có ít nhất hai xung nhiễu xuất hiện. Loại hazard này thường xảy ra trong các mạch tổ hợp.

Định nghĩa 5: Hazard nhất thời gọi là *hazard hàm số* trong thời gian chuyển đổi từ Q→P nếu:

- *f(Q)=f(P)*
- Hàm *f(X)* lấy cả hai giá trị 1 và 0 trong thời gian chuyển đổi từ Q→P

Định nghĩa 6: Hazard nhất thời gọi là *hazard logic* trong thời gian chuyển đổi từ Q→P nếu:

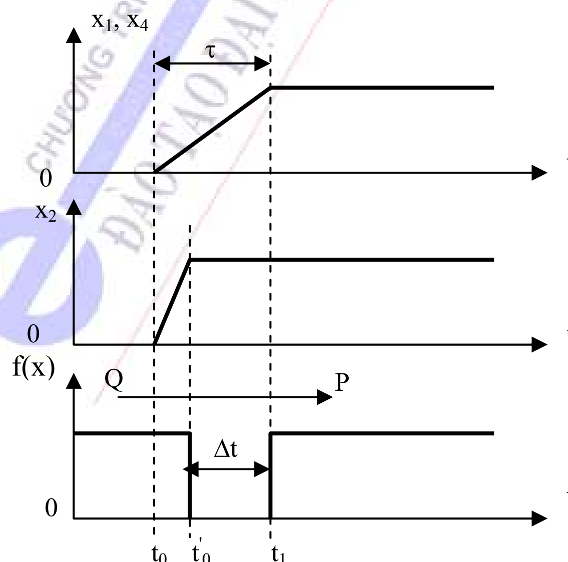
- *f(Q)=f(P)*
- Hàm *f(X)* chỉ nhận một giá trị như nhau (hoặc 0 hoặc 1)
- Trong thời gian chuyển đổi từ Q→P xuất hiện một xung hazard ở đầu ra.

4.4.3.1. Hazard tĩnh trong mạch logic.

Do có hiện tượng "chạy đua" giữa các tín hiệu vào với nhau trong thời gian chuyển từ Q→P mà xuất hiện hazard. Nếu f(Q) = f(P) tức là có sự thay đổi của tín hiệu vào nhưng sự điều khiển ở đầu ra của mạch logic vẫn không đổi dù là 0 hay 1, nhưng xuất hiện hazard, khi số lượng tín hiệu chạy đua không nhiều, đó chính là hazard tĩnh.

Hazard nhất thời cũng chính là hazard tĩnh, tức là loại hazard chỉ xuất hiện như một xung không theo quy định của hàm logic. Hiện tượng này không nguy hiểm, vì độ rộng của xung hazard tĩnh Δt luôn nhỏ hơn thời gian trễ τ của mạch, nên mạch logic vẫn hoạt động bình thường dù có xuất hiện hazard.

Nhưng hazard tĩnh nguy hiểm ở chỗ: nó có thể gây ra "sai nhầm" cho điều khiển của hệ thống logic khi giá trị độ rộng hazard (Δt) đủ lớn, điều này sẽ xảy ra khi sự "chạy đua" của tín hiệu vào quá chênh lệch, nghĩa là có tín hiệu vào "chạy" quá nhanh còn tín hiệu khác lại "chạy" quá chậm, hiện tượng này được minh hoạ ở hình 4-6.



Hình 4-6. Chạy đua ở hazard tĩnh

Ta thấy x_2 trong quá trình "chạy đua" (thay đổi giá trị logic) đã "chạy" nhanh hơn so với tín hiệu x_1, x_4, thể hiện ở hình vẽ độ dốc xung x_2 lớn hơn, điều đó làm cho Δt của xung hazard tăng theo, khi đó xung hazard trở nên "nguy hiểm" hơn vì nó có thể kích lật chuyển một mạch điện tiếp sau hệ thống mạch logic, gây hiện tượng điều khiển "sai nhầm" trong mạch logic.

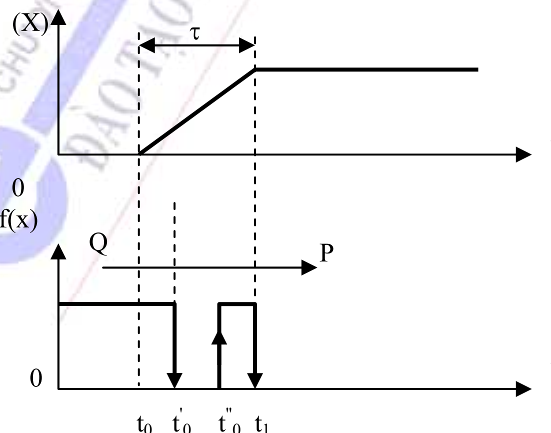
4.4.3.2. Hazard động trong mạch logic.

Trong thực tế khi thay đổi tín hiệu vào của mạch logic ứng với quá trình chuyển đổi ($Q \rightarrow P$) có thể có rất nhiều tín hiệu vào cùng thay đổi khi đó có sự chạy đua của các tín hiệu vào tới đầu ra của mạch. Ví dụ trường hợp Q = (0000); P = (1101), dễ dàng nhận thấy có sự chạy đua (X)

(X)	—	(x_1	x_2	x_3	x_4)	
t_0	—	0	0	0	0	f(Q) = 1
						↓
t'_0	—	0	1	0	0	f(X') = 0
						↓
t''_0	—	1	1	0	0	f(X'') = 1
						↓
t_1	—	1	1	0	1	f(P) = 0

Do có nhiều tín hiệu vào đồng thời thay đổi giá trị logic từ 0 sang 1 và từ 1 về 0 mà mỗi tín hiệu vào có tốc độ "chạy" khác nhau nên vô tình làm cho giá trị hàm f(X) ở đầu ra thay đổi như ở hình bên. Hiện tượng tín hiệu ra f(X) thay đổi giá trị từ $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ được gọi là hazard động, tức là xuất hiện nhiều xung không cần thiết trong khoảng thời gian trễ của mạch (τ). Như vậy trong thời gian rất nhỏ τ xuất hiện rất nhiều xung hazard nhỏ hơn τ thì ta có thể hiểu là xung hazard động không có gì nguy hiểm cả, vì một xung bị chia ra nhiều xung con thì năng lượng còn rất nhỏ và độ rộng xung quá bé nên không đủ kích mạch khác được. Hiện tượng này ta có thể hiểu là khi đèn dang sáng ta cho tín hiệu thay đổi để đèn tắt nhưng do có hiện tượng chạy đua nên sau khi đèn tắt thì lại hơi sáng lên rồi mới tắt hẳn.

Hazard động ít có khả năng gây ra điều khiển "sai nhầm" trong mạch logic tổ hợp.



Hình 4-7. Hazard động

4.4.3.3. Hazard hàm số trong mạch logic.

Hazard có thể xuất hiện do chức năng của mạch trong cả hai trường hợp là hàm f(X) lấy giá trị logic là 0 hoặc 1.

Hazard nhất thời gọi là *hazard hàm số* trong thời gian chuyển đổi từ Q→P nếu:

- *f(Q)=f(P)*

- Hàm *f(X)* lấy cả hai giá trị 1 và 0 trong thời gian chuyển đổi từ Q→P

Điều này có nghĩa là trong thời gian chuyển đổi Q→P thì hàm logic không thay đổi giá trị (f(Q)=f(P)), nhưng nếu lấy f(Q)=f(P) = 0 thì thì hazard vẫn xuất hiện hoặc lấy f(Q)=f(P)=1 thì hazard vẫn xảy ra. Hiện tượng này được gọi là hazard hàm số. Trên thực tế có những hàm số hazard nhất thời chỉ xuất hiện khi điều khiển logic là 1 (f(X) = 1) còn điều khiển logic ở đầu ra là 0 thì không có hazard nhất thời xuất hiện và ngược lại có thể điều khiển ra không bị hazard.

Độ nguy hiểm của hazard hàm số cũng giống như hazard tĩnh, nhưng nó nguy hiểm hơn một mức nữa vì bất kỳ quá trình điều khiển nào (0 hay 1) đều có khả năng xuất hiện hazard, tức là điều có khả năng gây ra "sai nhầm" khi điều khiển mạch.

4.4.3.4. Hazard logic trong mạch logic.

Đây là loại hazard nguy hiểm nhất, hay gây ra điều khiển "sai nhầm" nhiều nhất trong các hệ thống mạch tổ hợp điều khiển.

Bản chất của loại hazard này như sau:

Khi tập tín hiệu vào của hàm logic thay đổi đồng thời nhiều biến trong thời gian chuyển đổi Q → P, mà mỗi một lần tín hiệu vào có thời gian trễ khác nhau, trong quá trình "chạy đua" này gặp phải trường hợp Q = (00000), P = (11101)

(X)	—	(x_1	x_2	x_3	x_4	x_5)	
t_0	—	0	0	0	0	0	f(Q) = 1
				↓			↓
t'_0	—	0	0	1	0	0	f(X') = 0
			↓				↓
t''_0	—	0	1	1	0	0	f(X'') = 0
						↓	↓
t'''_0	—	0	1	1	0	1	f(X''') = 0
		↓					↓
t_1	—	1	1	1	0	1	f(P) = 1

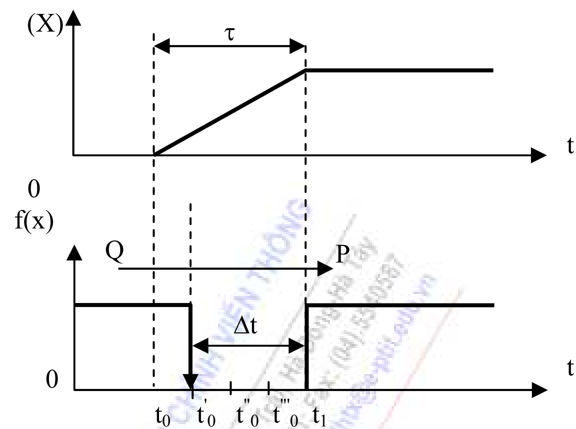
Hiện tượng hazard logic được mô tả trên hình 4-8:

Hazard nhất thời gọi là *hazard logic* trong thời gian chuyển đổi từ Q→P nếu:

- *f(Q)=f(P)*

- Hàm *f(X)* chỉ nhận một giá trị như nhau (hoặc 0 hoặc 1)

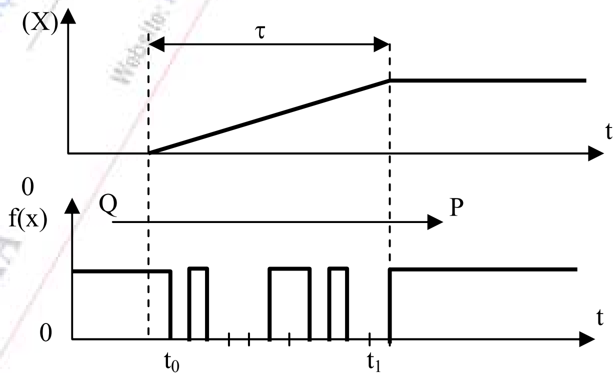
- Trong thời gian chuyển đổi từ Q→P xuất hiện một xung hazard có độ rộng Δt lớn ở đầu ra, khi quá trình chạy đua ngẫu nhiên của các tín hiệu vào tạo ra hàm f(X) có cùng một giá trị logic.



Hình 4-8. Hazard logic

Như vậy trong quá trình chuyển đổi từ Q → P của tập tín hiệu vào, có nhiều tín hiệu cùng thay đổi giá trị và hàm logic vô tình hay ngẫu nhiên xảy ra trường hợp có cùng một giá trị logic hazard ở đầu ra f(X) của mạch. Điều đó tạo nên một xung hazard ở đầu ra của của mạch độ rộng Δt lớn lên rất nhiều, khi Δt lớn làm cho xung hazard có năng lượng lớn đủ khả năng kích chuyển một mạch tiếp theo sau mạch điều khiển, điều đó gây ra hiện tượng điều khiển "sai nhầm" trong hệ thống logic tổ hợp. Đây là điều vô cùng nguy hiểm đối với các hệ thống tổ hợp cỡ lớn có nhiều đầu vào.

Trên thực tế quá trình chuyển đổi từ Q →P trong mạch logic tổ hợp rất phức tạp, rất ít khi gặp từng loại hazard riêng biệt mà gặp sự tổ hợp hỗn loạn các loại hazard trên. Hiện tượng này được minh hoạ bằng hình 4-9.

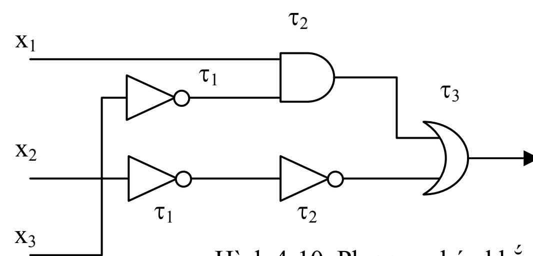


Hình 4-9. Hiện tượng tổng quát xuất hiện Hazard

Tóm lại, mỗi một mạch điều khiển có thể xuất hiện nhiều loại hazard, có mạch logic có số lượng biến số "chạy đua" rất lớn nhưng hazard lại không xuất hiện, nhưng có mạch rất đơn giản thì hazard lại xuất hiện và gây ra điều khiển "sai nhầm". Vì vậy muốn khắc phục được hazard thì phải căn cứ vào mạch điện cụ thể của nó, rồi dùng kỹ thuật phân tích phát hiện khả năng xuất hiện hazard, sau đó tìm cách khắc phục hazard. Sau đây là một vài biện pháp khắc phục và hạn chế sự xuất hiện hazard trong hệ thống logic diều khiển.

4.4.4. Các biện pháp khắc phục Hazard.

Như đã phân tích ở trên, hazard xuất hiện do có sự chạy đua tín hiệu vào trong hệ logic tổ hợp, nói cách khác hazard xuất hiện là do sự khác nhau về thời gian trễ truyền lan từ đầu vào đến đầu ra của mạch, từ đó ta có những biện pháp khắc phục hazard như sau:



Hình 4-10. Phương pháp khắc phục Hazard

- Biện pháp đơn giản nhất làm biến mất hazard là không để xuất hiện quá trình chạy đua của các tín hiệu vào trong mạch logic, nghĩa là chỉ thay đổi giá trị logic trên một đầu vào tín hiệu. Khi chỉ có một tín hiệu vào "chạy" trong mạch logic thì sẽ không còn "đua" tín hiệu nữa và chắc chắn hazard không thể xuất hiện. Nhưng như vậy cũng có nghĩa là từng tín hiệu vào thay đổi giá trị logic sẽ làm cho mạch hoạt động châm chạp, và không phải quá trình điều khiển nào cũng cho phép làm như vậy, thông thường có sự thay đổi nhiều tín hiệu vào cùng một lúc.

- Tiếp theo khi phải chấp nhận quá trình chuyển đổi từ Q→P có nhiều tín hiệu thay đổi hay có nhiều biến (X) chạy đua. Cách khắc phục là chọn giá trị linh kiện hay IC có thời gian trễ τ nhỏ. Vì ta biết hazard chỉ xuất hiện trong thời gian trễ của mạch, τ càng nhỏ nghĩa là xung hazard có độ rộng Δt nhỏ, và như vậy nó không có đủ năng lượng để kích chuyển mạch tiếp theo.Nhưng khi chọn linh kiện lắp ráp hệ thống hay chon IC có τ nhỏ tức là phải chọn linh kiện, IC có chất lượng cao, nghĩa là giá thành của hệ điều hành tăng, đây cũng là vấn đề cần quan tâm khi thiết mạch.

- Khi ta chấp nhận có sự chạy đua tín hiệu vào (X) trong quá trình chuyển đổi từ Q→P, đồng thời không dùng linh kiện có chất lượng cao để giảm giá thành và mạch vẫn hoạt động tốt đồng thời không có hazard xuất hiện, thì ta có thể dùng phương pháp khắc phục hazard bằng cách thêm các mạch trễ trên đường truyền tín hiệu, để đảm bảo cho thời gian chạy đua của các tín hiệu là tương đương nhau. Phương pháp này được minh hoạ ở hình 4-10:

Ta biết tín hiệu x_2 chạy nhanh tới đầu ra, nên trên đường truyền của x_2 ta cho thêm hai cổng đảo có thời gian trễ là τ_1 và τ_2 để cho tín hiệu trên x_2 xuất hiện đồng thời với x_1 và x_3, khi đó hazard sẽ không xuất hiện hoặc sẽ làm giảm bớt hazard . Phương pháp này có gây ra hazard nếu đường trễ thêm vào lại làm cho x_2 chạy quá chậm và lại phát sinh hiện tượng chạy đua tín hiệu vào.

Để tránh xảy ra hiện tượng chạy đua tín hiệu vào, cần biết chính xác thời gian trễ τ_1 và τ_2, sau đó phải tạo ra được cổng đảo có thời gian trễ bằng đúng giá trị τ_1 và τ_2.

- Ở mức cao hơn khi ta phải chấp nhận có sự chạy đua tín hiệu vào trong quá trình chuyển đổi Q →P, không muốn dùng linh kiện có chất lượng cao, đồng thời đã thêm các mạch trễ (không ảnh hưởng tới chức năng của mạch logic) nhưng vẫn không thể khắc phục hết hazard thì khi đó ta dùng xung đồng bộ, tức là ta bất chấp có sự chạy đua của tín hiệu vào, và giữa các đường truyền tín hiệu từ đầu vào tới đầu ra có thời gian trễ khác nhau. Nhưng tín hiệu truyền lan trong hệ logic dù nhanh, dù chậm, đến trước hay đến sau thì chúng chỉ được lan truyền khi có sự cho phép của xung đồng bộ. Xung đồng bộ thông thường "chờ" theo đường tín hiệu chạy chậm nhất, khi đó các xung đến sớm phải "chờ" cho đầy đủ các tín hiệu khác khi đó xung đồng bộ mới cho phép truyền tiếp. Nếu cho thêm vào mạch điều khiển xung đồng bộ thì cũng có thể giảm đáng kể ảnh hưởng của hazard.

- Trong trường hợp các phương pháp nêu trên đều được áp dụng nhưng hiện tượng hazard vẫn xuất hiện thì ta buộc phải thay đổi chức năng điều khiển, tức là thay đổi chức năng của hàm logic của hệ thống điều khiển tức là phải xây dựng mạch điện khác.

Như vậy để có được một mạch điều khiển tốt, chất lượng cao thì phần cứng xây dựng nên mạch điện mang tính quyết định. Người thiết kế phải hiểu rất kỹ và sâu sắc hệ thống kỹ thuật mà mình thiết kế thì mới có thể khắc phục được hazard trong mạch điện, cũng như phải biết thêm hay bớt các mạch điện phụ như thế nào mà không làm thay đổi chức năng của hệ thống. Từ đó làm

cho mạch có chất lượng cao hơn, giá trị kinh tế cũng cao hơn. Điều này cung dễ hiểu là các mạch điện có cùng chức năng điều khiển nhưng mỗi hãng sản xuất lại đưa ra một mạch khác nhau và giá trị kinh tế của chúng cũng khác nhau, tuỳ thuộc vào trình độ và sự quan tâm đến việc tăng độ tin cậy, tăng chất lượng điều khiển mạch của hãng. Nhưng bản chất vẫn chỉ là làm giảm tối đa khả năng xuất hiện hazard trong mạch.

4.5. MẠCH MÃ HOÁ VÀ GIẢI MÃ

4.5.1. Một số loại mã thông dụng.

4.5.1.1. Mã BCD và mã dư 3.

MÃ BCD (Binary Coded Decimal) là mã được cấu tạo bằng cách dùng từ nhị phân 4 bit để mã hóa 10 kí hiệu thập phân, nhưng cách biểu diễn vẫn theo thập phân. Ví dụ đối với mã NBCD, các chữ số thập phân được nhị phân hoá theo trọng số như nhau 2^3, 2^2, 2^1, 2^0 nên có 6 tổ hợp dư, ứng với các số thập phân 10,11,12,13,14 và 15. Sự xuất hiện các tổ hợp này trong bản tin được gọi là lỗi dư.

Do trọng số nhị phân của mỗi vị trí biểu diễn thập phân là tự nhiên nên máy có thể thực hiện trực tiếp các phép tính cộng, trừ, nhân, chia theo mã NBCD. Tuy nhiên nhược điểm chính của mã là tồn tại tổ hợp toàn Zero, gây khó khăn trong việc đồng bộ khi truyền dẫn tín hiệu.

Vì vậy, người ta sử dụng mã Dư-3 được hình thành từ mã NBCD bằng cách cộng thêm 3 vào mỗi tổ hợp mã. Như vậy, mã không bao gồm tổ hợp toàn Zero. Mã Dư-3 chủ yếu được dùng để truyền dẫn tín hiệu mà không dùng cho việc tính toán trực tiếp.

Thập phân	BCD 8421	Mã dư 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Bảng 4-2. Mã BCD 8421 và mã dư 3

4.5.1.2. Mã Gray.

Mã Gray còn được gọi là mã cách 1, là loại mã mà các tổ hợp mã kế nhau chỉ khác nhau duy nhất 1 bit. Loại mã này không có tính trọng số. Do đó, giá trị thập phân đã được mã hóa chỉ được giải mã thông qua bảng mã mà không thể tính theo tổng trọng số như đối với mã BCD.

Mã Gray có thể được tổ chức theo nhiều bit. Bởi vậy, có thể đếm theo mã Gray.

Cũng tương tự như mã BCD, ngoài mã Gray chính còn có mã Gray dư-3.

Thập phân	Gray	Gray dư 3
0	0000	0010
1	0001	0110
2	0011	0111
3	0010	0101
4	0110	0100
5	0111	1100
6	0101	1101
7	0100	1111
8	1100	1110
9	1101	1010
10	1111	1011
11	1110	1001
12	1010	1000
13	1011	0000
14	1001	0001
15	1000	0011

Bảng 4-3. Mã Gray và Gray dư 3

4.5.1.3. Mã chẵn, lẻ.

Mã chẵn và mã lẻ là hai loại mã có khả năng phát hiện lỗi hay dùng nhất. Để thiết lập loại mã này ta chỉ cần thêm một bit chẵn/ lẻ (bit parity) vào tổ hợp mã đã cho, nếu tổng số bit 1 trong từ mã (bit tin tức + bit chẵn/lẻ) là chẵn thì ta được mã chẵn và ngược lại ta được mã lẻ.

BCD 8421	BCD 8421chẵn	P_C	BCD 8421lẻ	P_L
0000	0000	0	0000	1
0001	0001	1	0001	0
0010	0010	1	0010	0
0011	0011	0	0011	1
0100	0100	1	0100	0
0101	0101	0	0101	1
0110	0110	0	0110	1
0111	0111	1	0111	0
1000	1000	1	1000	0
1001	1001	0	1001	1

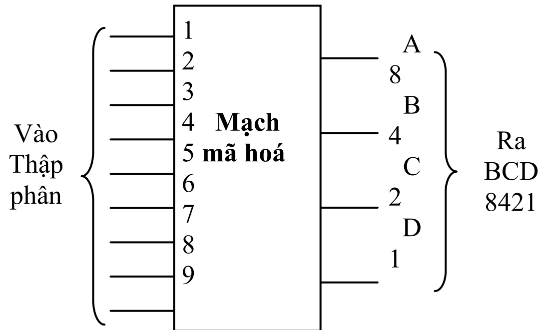
Bảng 4-4. Mã BCD 8421 chẵn / lẻ

4.5.2. Mạch mã hoá.

Mạch điện thực hiện việc chuyển tin tức sang mã, được gọi là mạch mã hoá hay mạch ghi mã.

4.5.1.1. Mạch mã hoá từ thập phân sang BCD 8421

Sơ đồ khối tổng quát của mạch Mã hoá như hình 4-7. Mạch gồm 9 lối vào (biến) ứng Với các chữ số thập phân từ 1 đến 9. Lối vào zero là không cần thiết, vì khi tất cả các lối vào khác bằng 0 thì lối ra cũng bằng 0. Bốn lối ra A, B, C, D (hàm) thể hiện tổ hợp mã tương ứng với mỗi chữ số thập phân trên lối vào theo trọng số 8421. Bảng trạng thái của mạch như bảng 4-5.



Hình 4-11 Sơ đồ khối của mạch mã hoá

Vào thập phân	Ra BCD 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Bảng 4-5. Bảng trạng thái của mạch mã hoá.

Từ bảng trạng thái ta viết được các hàm ra như sau:

$A = 8 + 9 \quad = \Sigma\,(8,9)$

$B = 4 + 5 + 6 + 7 \quad = \Sigma\,(4,5,6,7)$

$C = 2 + 3 + 6 + 7 \quad = \Sigma\,(2,3,6,7)$

$D = 1 + 3 + 5 + 7 + 9 \quad = \Sigma\,(1,3,5,7,9)$

Căn cứ hệ phương trình, ta xây dựng được mạch điện của bộ mã hoá. Hoặc dùng ma trận diode (cổng OR) để xây dựng

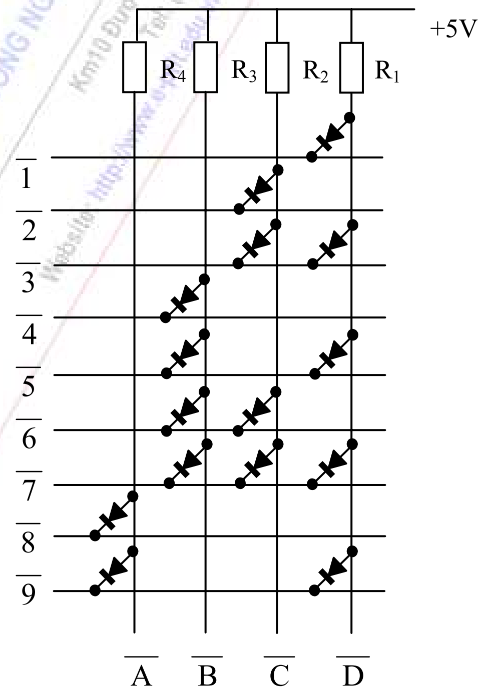
Hoặc có thể được viết lại như sau (dùng định lý DeMorgan) và dùng ma trận diode (cổng AND) để xây dựng mạch:

$\overline{A} = \overline{8+9} = \overline{8}\,.\,\overline{9}$

$\overline{B} = \overline{4+5+6+7} = \overline{4}\,.\,\overline{5}\,.\,\overline{6}\,.\,\overline{7}$

$\overline{C} = \overline{2+3+6+7} = \overline{2}\,.\,\overline{3}\,.\,\overline{6}\,.\,\overline{7}$

$\overline{D} = \overline{1+3+5+7+9} = \overline{1}\,.\,\overline{3}\,.\,\overline{5}\,.\,\overline{7}\,.\,\overline{9}$



Hình 4-12 Mạch điện của bộ mã hoá dùng diode.

4.5.1.2. Mạch mã hoá ưu tiên

Trong bộ mã hoá vừa xét trên, tín hiệu vào tồn tại độc lập, (không có trường hợp có 2 tổ hợp trở lên đồng thời tác động). Bộ mã hoá ưu tiên ra đời để giải quyết trường hợp có nhiều đầu vào tác động đồng thời. Đối với các trường hợp này thì bộ mã hoá ưu tiên chỉ tiến hành mã hoá tín hiệu vào nào có cấp ưu tiên cao nhất ở thời điểm xét. Việc xác định cấp ưu tiên cho mỗi tín hiệu vào là do người thiết kế mạch.

Bây giờ ta xét nguyên tắc hoạt động và quá trình thiết kế của bộ mã hoá ưu tiên 9 lối vào, 4 lối ra.

Vào Thập phân $L_1L_2L_3L_4L_5L_6L_7L_8L_9$	**Ra** A B C D 8 4 2 1
0 0 0 0 0 0 0 0 0	0 0 0 0
1 0 0 0 0 0 0 0 0	0 0 0 1
x 1 0 0 0 0 0 0 0	0 0 1 0
x x 1 0 0 0 0 0 0	0 0 1 1
x x x 1 0 0 0 0 0	0 1 0 0
x x x x 1 0 0 0 0	0 1 0 1
x x x x x 1 0 0 0	0 1 1 0
x x x x x x 1 0 0	0 1 1 1
x x x x x x x 1 0	1 0 0 0
x x x x x x x x 1	1 0 0 1

Bảng 4-6. Bảng trạng thái của bộ mã hoá ưu tiên

Theo đề bài, sự mã hoá thực hiện theo mức độ ưu tiên từ L_1 đến L_9, khi các tín hiệu cùng tác động thì các tín hiệu có mức ưu tiên thấp không tác dụng, nghĩa là bất kể mức logic của nó là 0 hay 1 đều không ảnh hưởng đến lối ra nên gọi nó là điều kiện tuỳ chọn, ký hiệu là "x".

Bảng trạng thái phản ánh yêu cầu thiết kế, mã hoá theo cấp ưu tiên.

Từ bảng trạng thái ta có thể viết được biểu thức lối ra như sau:

D = 1 tại các lối: + L_1 và bằng 0 tại các lối L_2, L_4, L_6, L_8

+ L_3 và bằng 0 tại các lối L_4, L_6, L_8

+ L_5 và bằng 0 tại các lối L_6, L_8

+ L_7 và bằng 0 tại các lối L_8

+ L_9

Nên ta viết được hàm D:

$$D = L_1.\overline{L_2}.\overline{L_4}.\overline{L_6}.\overline{L_8} + L_3.\overline{L_4}.\overline{L_6}.\overline{L_8} + L_5.\overline{L_4}.\overline{L_6}.\overline{L_8} + L_7.\overline{L_8} + L_9$$

Tương tự như vậy ta viết được hàm của B, C và A như sau:

$$C = L_2.\overline{L_4}.\overline{L_5}.\overline{L_8}.\overline{L_9} + L_3.\overline{L_4}.\overline{L_5}.\overline{L_8}.\overline{L_9} + L_6.\overline{L_8}.\overline{L_9} + L_7.\overline{L_8}.\overline{L_9}$$

$$B = L_4.\overline{L_8}.\overline{L_9} + L_5.\overline{L_8}.\overline{L_9} + L_6.\overline{L_8}.\overline{L_9} + L_7.\overline{L_8}.\overline{L_9}$$

$$A = L_8.L_9$$

Một vài IC thường dùng: 74147 là bộ mã hoá ưu tiên NBCD 4 bit, 74148 là bộ mã hoá ưu tiên NBCD 3 bit.

4.5.2. Bộ giải mã.

Mạch điện thực hiện việc chuyển từ mã sang tin tức được gọi là mạch giải mã hoá.

4.5.2.1. Bộ giải mã nhị phân

Bộ giải mã nhị phân còn có tên là bộ giải mã "*1* từ *n*", bộ giải mã địa chỉ hoặc bộ chọn địa chỉ nhị phân. Chức năng của nó là lựa chọn duy nhất một lối ra (lấy giá trị 1 hoặc 0), khi tác động tới đầu vào một số nhị phân.

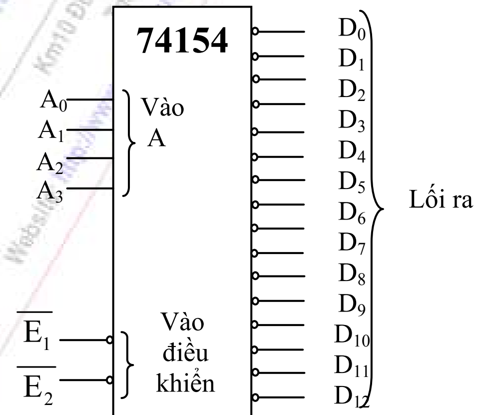
Như vậy, nếu số nhị phân là *n* bit (*n* lối vào) sẽ nhận diện được 2^n địa chỉ khác nhau (trên 2^n lối ra). Nói khác đi, mạch chọn địa chỉ nhị phân là một mạch logic tổ hợp có *n* lối vào và 2^n lối ra, nếu tác động tới đầu vào một số nhị phân thì chỉ duy nhất một lối ra được lựa chọn, lấy giá trị 1 (tích cực cao) hoặc 0 (tích cực thấp), các lối ra còn lại đều không được lựa chọn, lấy giá trị 0 hoặc 1. Sơ đồ khối tổng quát của bộ chọn địa chỉ nhị phân như chỉ ở hình 4-13.



Hình 4-13. Sơ đồ khối của bộ giải mã nhị phân

IC 74154 là một bộ chọn địa chỉ nhị phân 4 vào 16 ra. Ký hiệu logic của nó được chỉ ra ở hình 4-14. Các lối vào E_1, E_2, hoạt động theo tích cực thấp thường được sử dụng để mở rộng dung lượng hoặc thay đổi chức năng logic của bộ chọn địa chỉ.

Ta có thể mở rộng dung lượng bộ chọn địa chỉ nhị phân bằng cách ghép các IC có dung lượng nhỏ lại với nhau.

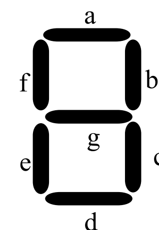


Hình 4-14. Ký hiệu logic của IC 74154

4.5.2.2. Mạch giải mã 7 đoạn

a) Dụng cụ 7 đoạn

Để hiển thị chữ số của một hệ đếm phân bất kỳ, ta có thể dùng dụng cụ 7 đoạn. Cấu tạo của nó như chỉ ở hình 4-15. Các đoạn được hình thành bằng nhiều loại vật liệu khác nhau, nhưng phải có khả năng hiển thị được trong các điều kiện ánh sáng khác nhau và tốc độ chuyển mạch phải đủ lớn. Trong kĩ thuật số, các đoạn thường được dùng là LED hoặc tinh thể lỏng (LCD).



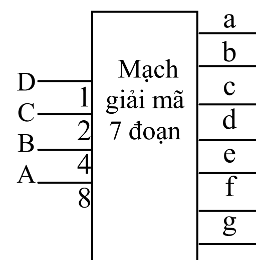
Hình 4-15 Cấu tạo dụng cụ 7 đoạn sáng

Đối với LED, mỗi đoạn là một Diode phát quang và khi có dòng điện đi qua đủ lớn (5 đến 30 mA) thì đoạn tương ứng sẽ sáng.

Ngoài 7 đoạn sáng chính, mỗi LED cũng có thêm Diode để hiển thị dấu phân số khi cần thiết. LED có hai loại chính: LED Anôt chung và Ktốt chung. Do đó, logic của tín hiệu điều khiển hai loại này là ngược nhau.

b) Mạch giải mã 7 đoạn

Nhiệm vụ của ta là phải thiết kế một mạch logic liên hợp với 4 lối vào và 7 lối ra để chuyển mã NBCD thành mã 7 đoạn. Sơ đồ khối tổng quát của bộ giải mã như hình 4-16. Từ hình 4-15 dễ



Hình 4-16 Sơ đồ khối của mạch giải mã 7 đoạn sáng

nhận thấy rằng, đoạn *a* sẽ sáng khi hiển thị chữ số : 0 hoặc 2, hoặc 3, hoặc 5, hoặc 7, hoặc 8, hoặc 9. Do đó, ta có thể viết:

$a = \sum(0,2,3,5,6,7,8,9)$. Tương tự, ta có:

$b = \sum(0,1,2,3,4,7,8,9)$,

$c = \sum(0,1,3,4,5,6,7,8,9)$,

$d = \sum(0,2,3,5,6,8,9)$,

$e = \sum(0,2,6,8)$,

$f = \sum(0,4,5,6,8,9)$,

$g = \sum(2,3,4,5,6,8,9)$.

IC 7447, 74247 (Anốt chung), 7448 (K chung), 4511 (CMOS) là các IC giải mã từ NBCD sang thập phân theo phương pháp hiển thị 7 đoạn.

4.6 BỘ HỢP KÊNH VÀ PHÂN KÊNH

4.6.1 Bộ hợp kênh (MUX-Multiplexer)

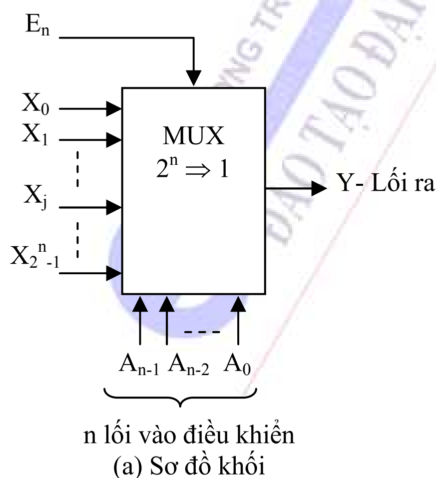
Định nghĩa: Bộ hợp kênh là mạch có 2^n lối vào dữ liệu, *n* lối vào điều khiển, 1 lối vào chọn mạch và 1 lối ra.

Tuỳ theo giá trị của *n* lối vào điều khiển mà lối ra sẽ bằng một trong những giá trị ở lối vào (X_j). Nếu giá trị thập phân của *n* lối vào điều khiển bằng *j* thì $Y = X_j$.

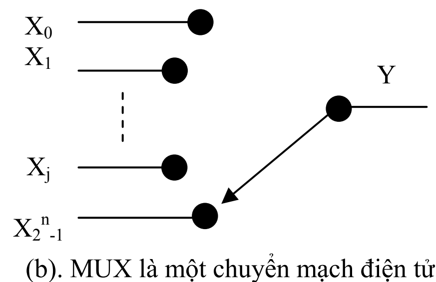
Sơ đồ khối của MUX $2^n \Rightarrow 1$ (2^n lối vào, 1 lối ra) được biểu diễn ở hình 4-17a.

Phương trình tín hiệu ra là:

$$Y = X_0(\overline{A_{n-1}}\,\overline{A_{n-2}}...\overline{A_i}...\overline{A_0}) + X_1(\overline{A_{n-1}}\,\overline{A_{n-2}}...\overline{A_i}...A_0) + ... + X_{2^n-1}(A_{n-1}A_{n-2}...A_i...A_1A_0)$$



(a) Sơ đồ khối



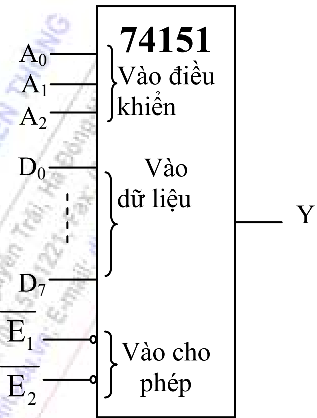
(b). MUX là một chuyển mạch điện tử

Hình 4-17. Bộ hợp kênh MUX $2^n \Rightarrow 1$

Thực chất, MUX là chuyển mạch điện tử dùng các tín hiệu điều khiển ($A_{n-1}A_{n-2}\ldots A_0$) để điều khiển sự nối mạch của lối ra với 1 trong số 2^n lối vào (hình 4-17b).

Hiện nay, bộ MUX được dùng như một phần tử vạn năng để xây dựng những mạch tổ hợp khác.

IC 74151 là bộ MUX 8 lối vào dữ liệu - 1 lối ra. Hình 4-18 là ký hiệu logic của IC 74151.



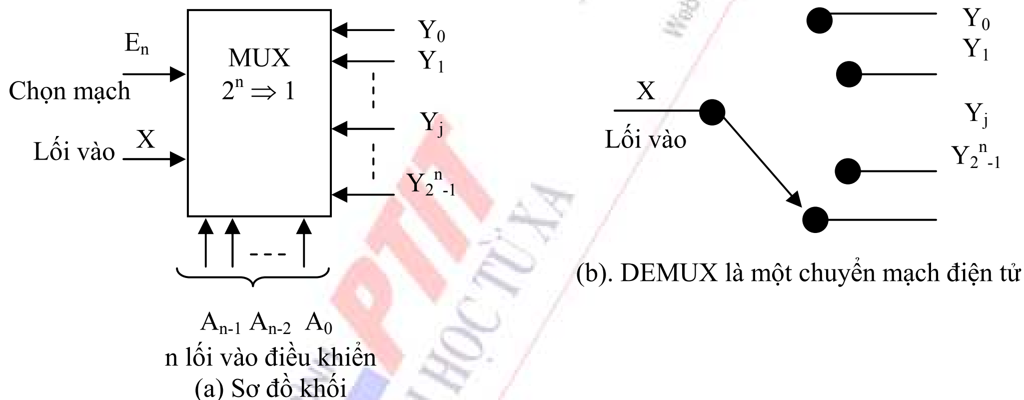
Hình 4-18. Ký hiệu logic của IC 74151

4.6.2 Bộ phân kênh (DEMUX-DeMultiplexer)

Định nghĩa: Bộ phân kênh là mạch có 1 lối vào dữ liệu, *n* lối vào điều khiển, 1 lối vào chọn mạch và 2^n lối ra.

Tuỳ theo giá trị của *n* lối vào điều khiển mà lối ra thứ *i* (Y_i) sẽ bằng giá trị của lối vào. Cụ thể nếu gọi *n* lối vào điều khiển là $A_{n-1}A_{n-2}\ldots A_0$ thì $Y_i = X$ khi $(A_{n-1}A_{n-2}\ldots A_1A_0)_2 = (i)_{10}$.

Sơ đồ khối của bộ DEMUX 1 lối vào 2^n lối ra được biểu diễn ở hình 4-19.



Hình 4-19. Bộ phân kênh DEMUX $1 \Rightarrow 2^n$

Phương trình tín hiệu ra của DEMUX $1 \Rightarrow 2^n$:

$$Y_0 = X.\overline{A}_{n-1}\overline{A}_{n-2}\ldots\overline{A}_i\ldots\overline{A}_0$$

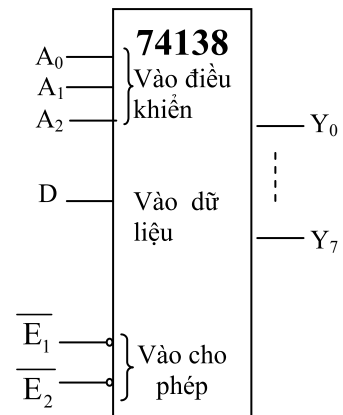
$$Y_1 = X.\overline{A}_{n-1}\overline{A}_{n-2}\ldots\overline{A}_i\ldots\overline{A}_1A_0$$

....

$$Y_{2^n-1} = X.A_{n-1}.A_{n-2}\ldots A_i\ldots A_0$$

Bộ phân kênh còn được gọi là bộ giải mã 1 trong 2^n. Tại một thời điểm chỉ có 1 trong số 2^n lối ra ở mức tích cực.

IC 74138 là bộ DEMUX 1 lối vào dữ liệu - 8 lối ra. Hình 4-20 là ký hiệu logic của IC 74138.



Hình 4-20. Ký hiệu logic của IC 74138

4.7. MẠCH CỘNG.

4.7.1. Mạch toàn tổng.

Mạch cộng hay (bộ cộng) là mạch số học nhị phân quan trọng, vì trong xử lý nhị phân phần lớn các phép tính được thực hiện thông qua phép cộng.

Mạch logic thực hiện phép cộng hai số nhị phân 1 bit có lối nhớ đầu vào được gọi là mạch toàn tổng. Sơ đồ khối tổng quát của một mạch toàn tổng được biểu diễn ở hình 4-21.

Theo hình 4-21 và nguyên lý cộng hai số nhị phân một bit có trọng số bất kỳ, ta có thể lập bảng trạng thái cho mạch toàn tổng.

Các hàm ra S_i , C_i sẽ có dạng:

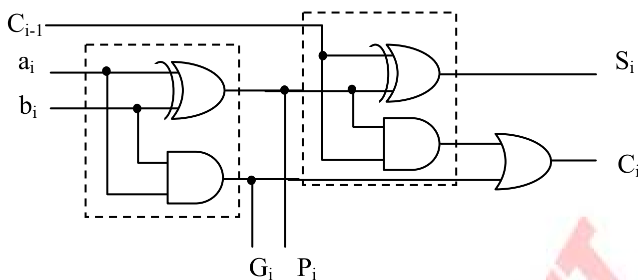
$$S_i = a_i \oplus b_i \oplus C_{i-1}$$

$$C_i = a_i b_i C_{i-1} + \overline{a_i}\, b_i\, C_{i-1} + a_i\, \overline{b_i}\, C_{i-1} \quad \text{hay}$$

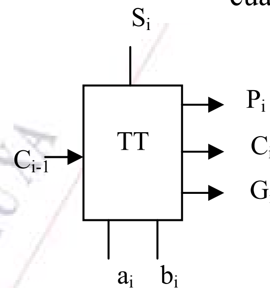
$$C_i = a_i b_i + (a_i \oplus b_i)\, C_{i-1}$$

C_{i-1}	a_i	b_i	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Bảng 4-7. Bảng trạng thái của mạch toàn tổng.



a) Mạch điện



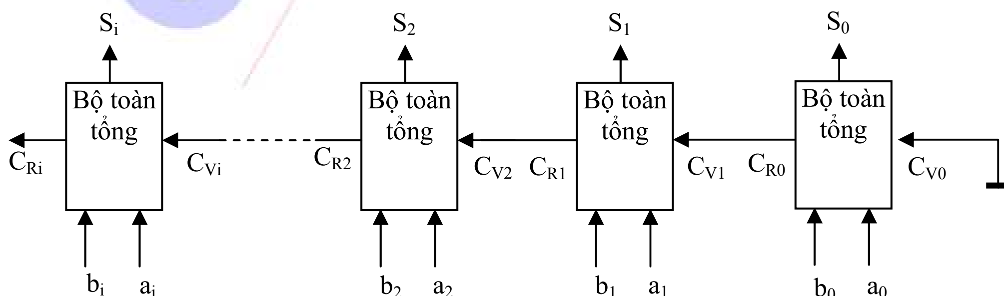
b) Ký hiệu

Hình 4-21 a, b Mạch toàn tổng và ký hiệu

Mạch logic thực hiện biểu thức lối ra tổng và lối ra nhớ được trình bày ở hình 4-21a và ký hiệu của nó là hình 4-21b.

4.7.2 Mạch cộng nhị phân song song

Ta có thể ghép nhiều bộ cộng hai số nhị một bit lại với nhau để thực hiện phép cộng hai số nhị phân nhiều bit. Sơ đồ khối của bộ cộng được trình bày ở hình 4-22 và được gọi là bộ cộng song song.



Hình 4-22 Sơ đồ khối của bộ cộng nhị phân song song

Để giảm bớt mức độ phức tạp của mạch, trong thực tế người ta thường sản xuất bộ tổng 4 bit. Muốn cộng nhiều bit, có thể hợp nối tiếp một vài bộ tổng một bit theo phương pháp nêu trên.

Một trong những bộ cộng thông dụng hiện nay là 7483. IC này được sản xuất theo hai loại: 7483 và 7483A với logic vào, ra khác nhau.

4.8. MẠCH SO SÁNH.

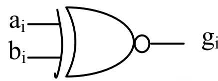
Trong các hệ thống số, đặc biệt là trong máy tính, thường thực hiện việc so sánh hai số. Hai số cần so sánh có thể là các số nhị phân, có thể là các ký tự đã mã hoá nhị phân. Mạch so sánh có thể hoạt động theo kiểu nối tiếp hoặc theo kiểu song song. Trong phần này ta sẽ nghiên cứu bộ so sánh theo kiểu song song.

4.8.1. Bộ so sánh bằng nhau.

4.8.1.1. Bộ so sánh bằng nhau 1 bit.

Xét 2 bit a_i và b_i, gọi g_i là kết quả so sánh. Từ đó là có bảng trạng thái 4-8.

$$g_i = \overline{a_i}.\overline{b_i} + a_i.b_i = \overline{a_i \oplus b_i}$$



Hình 4-23. Sơ đồ logic hàm ra của bộ so sánh bằng 1 bit

a_i	b_i	g_i
0	0	1
0	1	0
1	0	0
1	1	1

Bảng 4-8. Bảng trạng thái của mạch so sánh bằng.

4.8.1.2. Bộ so sánh bằng nhau 4 bit.

So sánh hai số nhị phân 4 bit A = $a_3a_2a_1a_0$ với B = $b_3b_2b_1b_0$. Vậy hai số A và B bằng nhau khi $a_3 = b_3$, $a_2 = b_2$, $a_1 = b_1$, $a_0 = b_0$.

Biểu thức đầu ra tương ứng là:

$$G = g_3.\ g_2.\ g_1.\ g_0$$

với

$$g_3 = \overline{a_3 \oplus b_3}$$
$$g_2 = \overline{a_2 \oplus b_2}$$
$$g_1 = \overline{a_1 \oplus b_1}$$
$$g_0 = \overline{a_0 \oplus b_0}$$

4.8.2. Bộ so sánh.

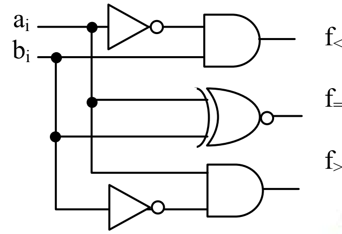
4.8.2.1. Bộ so sánh 1 bit.

Từ bảng trạng thái 4-9 ta có biểu thức ra:

$$f_< = \overline{a_i}.b_i$$
$$f_= = \overline{a_i \oplus b_i}$$
$$f_> = a_i.\overline{b_i}$$

a_i	b_i	$f_<$	$f_=$	$f_>$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Bảng 4-9. Bảng trạng thái của mach so sánh.



Hình 4-24. Mạch điện của bộ so sánh 1 bit

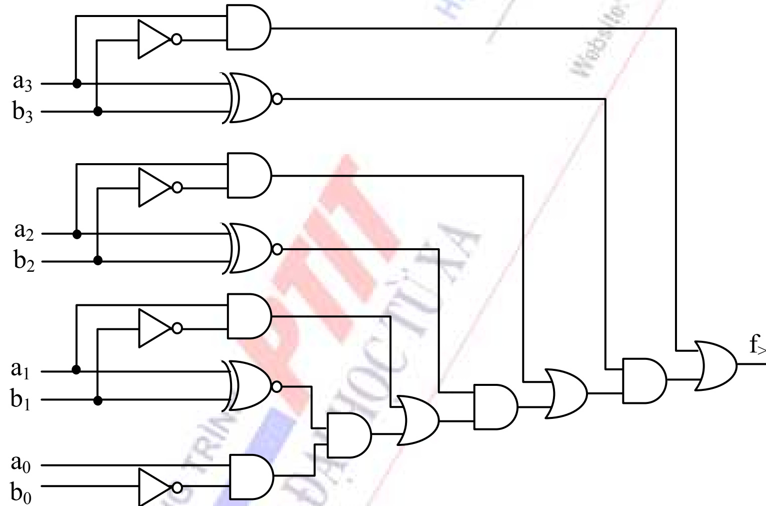
4.8.2.2. Bộ so sánh 4 bit (So sánh lớn hơn).

So sánh hai số nhị phân 4 bit A = $a_3a_2a_1a_0$ với B = $b_3b_2b_1b_0$. Số A lớn hơn số B khi:

$a_3 > b_3$ hoặc $a_3 = b_3$ và $a_2 > b_2$ hoặc $a_3 = b_3$ và $a_2 = b_2$ và $a_1 > b_1$ hoặc $a_3 = b_3$ và $a_2 = b_2$ và $a_1 = b_1$ và $a_0 > b_0$.

Từ đó ta có biểu thức hàm ra là:

$$f_> = a_3.\overline{b_3} + \overline{a_3 \oplus b_3}.a_2.\overline{b_2} + \overline{a_3 \oplus b_3}.\overline{a_2 \oplus b_2}.a_1.\overline{b_1} + \overline{a_3 \oplus b_3}.\overline{a_2 \oplus b_2}.\overline{a_1 \oplus b_1}.a_0.\overline{b_0}$$



Hình 4-26. Mạch điện của bộ so sánh lớn hơn 4 bit

Một trong những bộ so sánh thông dụng hiện nay là 7485. IC này so sánh 2 số nhị phân 4 bit.

4.9. MẠCH TẠO VÀ KIỂM TRA CHẴN LẺ.

Có nhiều phương pháp mã hoá dữ liệu để phát hiện lỗi và sửa lỗi khi truyền dữ liệu từ nơi này sang nơi khác. Phương pháp đơn giản nhất là thêm một bit vào dữ liệu được truyền đi sao cho số chữ số 1 trong dữ liệu luôn là chẵn hoặc lẻ. Bit thêm vào đó được gọi là bit chẵn/lẻ.

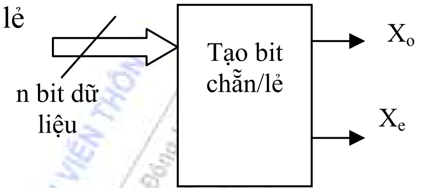
Để thực hiện được việc truyền dữ liệu theo kiểu đưa thêm bit chẵn, lẻ vào dữ liệu chúng ta phải:

- Xây dựng sơ đồ tạo được bit chẵn, lẻ để thêm vào *n* bit dữ liệu.

- Xây dựng sơ đồ kiểm tra hệ xem đó là hệ chẵn hay lẻ với (n + 1) bit ở đầu vào (*n* bit dữ liệu, 1 bit chẵn/lẻ).

4.9.1. Mạch tạo bit chẵn/lẻ.

Gọi 3 bit của dữ liệu là d_1, d_2, d_3 và X_e, X_o là 2 bit chẵn, lẻ thêm vào dữ liệu. Từ đó lập được bảng trạng thái 4-10:



Hình 4-27. Sơ đồ khối của mạch tạo bit chẵn/lẻ

Vào			Ra	
d_1	d_2	d_3	X_e	X_o
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Bảng 4-10. Bảng trạng thái của mạch tạo bit chẵn lẻ

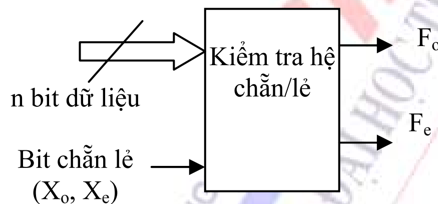
Từ bảng trạng thái ta thấy $X_o = \overline{X_e}$ hay $X_e = \overline{X_o}$.

Và biểu thức của X_o và X_e là

$X_o = d_1 \oplus d_2 \oplus d_3$

$X_o = \overline{X_e} = \overline{d_1 \oplus d_2 \oplus d_3}$

4.9.2. Mạch kiểm tra chẵn/lẻ.



Hình 4-28. Sơ đồ khối của mạch kiểm tra chẵn/lẻ

Vào				Ra	
d_1	d_2	d_3	X	F_e	Fo
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

Bảng 4-11. Bảng trạng thái của mạch kiểm tra chẵn /lẻ

Bảng trạng thái của mạch kiểm tra tính chẵn/lẻ của hệ được cho ở bảng 4-11.

Từ bảng trạng thái ta thấy:

- F_e = 1 nếu hệ là chẵn (F_e chỉ ra tính chẵn của hệ).
- F_o = 1 nếu hệ là lẻ (F_o chỉ ra tính lẻ của hệ).

Hai hàm này luôn là phủ định của nhau. Mặt khác do tính chất của hàm cộng XOR, ta có:

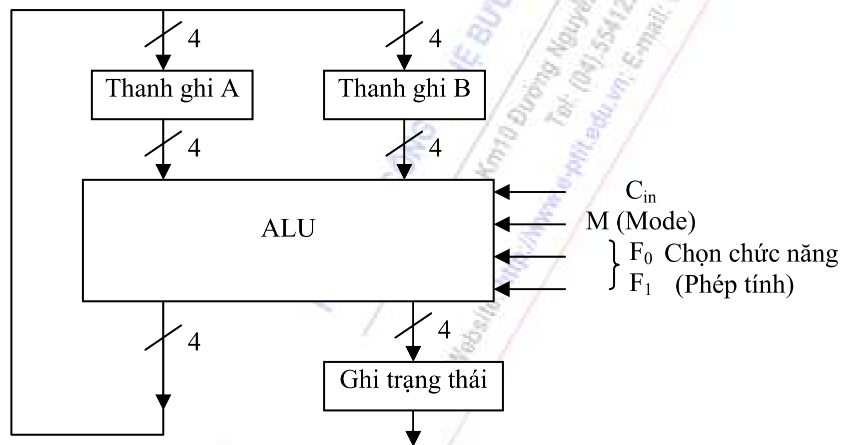
- $F_o = d_1 \oplus d_2 \oplus d_3 \oplus X$
- $F_o = \overline{F_e}$

4.10. ĐƠN VỊ SỐ HỌC VÀ LOGIC (ALU).

Đơn vị số học và logic (Arithmetic – Logic Unit) là một thành phần cơ bản không thể thiếu được trong các máy tính. Nó bao gồm 2 khối chính là khối logic và khối số học và một khối ghép kênh.

- Khối logic: Thực hiện các phép tính logic như là AND, OR, NOT, XOR.
- Khối số học: Thực hiện các phép tính số học như là: cộng, trừ, tăng 1, giảm 1.

Sơ đồ khối của 1 đơn vị số học – logic ALU 4 bit được mô tả ở hình 4-29:



Hình 4-29. Sơ đồ khối của ALU 4 bit

M là lối vào chọn phép tính số học hay logic. F_0, F_1 là hai lối vào chọn chức năng. Sau khi một phép tính số học hay logic được thực hiện thì kết quả sẽ được ghi lên 1 thanh ghi, ví dụ thanh ghi A. Kết quả này có thể được sử dụng để thực hiện phép tính sau. Bộ ALU còn tạo ra các bit trạng thái chuyển đổi thanh ghi. Ví dụ: Carry out: nếu có nhớ; Zero: nếu kết quả phép tính bằng 0.

TÓM TẮT

Trong chương này, chúng ta đã giới thiệu mạch logic tổ hợp. Mạch tổ hợp do các phần tử logic cơ bản cấu trúc nên. Đặc điểm của mạch tổ hợp là tín hiệu đầu ra ở thời điểm bất kỳ nào cũng chỉ phụ thuộc vào tín hiệu ở đầu vào ở thời điểm đó mà không liên quan đến trạng thái vốn có của mạch.

Mạch tổ hợp rất phong phú, ta không thể xem xét hết trong chương 4. Trọng tâm của chúng ta là nắm vững đặc điểm mạch tổ hợp và phương pháp chung khi thiết kế, phân tích mạch tổ hợp. Vì vậy, chúng ta đã giới thiệu một cách chọn lọc bộ mã hoá, bộ giải mã, bộ hợp kênh, phân kênh, mạch cộng, trừ, mạch so sánh…trong quá trình đó, ta đã xem xét phương pháp phân tích và thiết kế mạch tổ hợp.

Khi phân tích mạch tổ hợp đã cho, ta có thể viết ra hàm logic đầu ra cho từng cấp của sơ đồ, rồi tiến hành tối thiểu hoá hàm logic đó để biểu thị rõ mối quan hệ giữa đầu ra với đầu vào. Cần lưu ý thêm rằng phải xem xét đến hiện tượng Hazard- là hiện tượng chạy đua trong mạch logic và cách khắc phục hiện tượng này.

Việc tối thiểu hoá hàm logic rất quan trọng. Vì việc này làm cho mạch logic đơn giản, kinh tế. Chúng ta mong muốn mạch điện càng ít linh kiện càng tốt, số đầu vào của mạch cổng cũng không thể quá nhiều

CÂU HỎI ÔN TẬP

1. Mạch logic tổ hợp là mạch:

   a. Có tín hiệu ở đầu ra chỉ phụ thuộc vào tín hiệu ở đầu vào của mạch tại thời điểm đang xét.

   b. Không những tín hiệu ở đầu ra phụ thuộc vào tín hiệu ở đầu vào mà còn phụ thuộc vào trạng thái trong của mạch tại thời điểm đang xét.

   c. Cả hai phương án trên đều đúng.

   d. Không có phương án nào đúng.

2. Loại Hazard có trong mạch lôgic tổ hợp có thể là loại:

   a. Hazard chỉ xuất hiện 1 lần và không bao giờ gặp nữa.

   b. Hazard có thể xuất hiện nhiều lần.

   c. Hazard có thể do chức năng của mạch điện gây ra.

   d. Cả 3 phương án trên đều đúng.

3. Loại Hazard nào trong mạch logic tổ hợp là loại nguy hiểm nhất?

   a. Hazard tĩnh.

   b. Hazard động.

   c. Hazard logic.

   d. Không có phương án nào đúng.

4. Bộ mã hoá ưu tiên là bộ mã hoá cho phép mã hoá khi:

   a. Chỉ có một tín hiệu tác động vào.

   b. Chỉ hai tín hiệu tác động vào.

   c. Có hai tín hiệu trở lên đồng thời tác động vào.

   d. Cả 3 phương án trên đều đúng.

5. Bộ giải mã BCD sang thập phân làm nhiệm vụ biến đổi:

   a. đầu vào nhị phân thành đầu ra thập lục phân (hệ hexa).

   b. đầu vào thập phân thành mã BCD 8-4-2-1.

   c. đầu vào BCD 8-4-2-1 thành đầu ra thập phân tương ứng.

   d. Không có phương án nào đúng.

6. Dụng cụ hiển thị 7-đoạn:

   a. chỉ có thể chỉ thị các ký tự từ 0 đến 9.

b. chỉ có thể chỉ thị các ký tự từ A đến F.

c. chỉ có thể chỉ thị các ký tự từ 0 đến 9 và từ A đến F.

d. có thể được cấu tạo để chỉ thị các ký hiệu số, chữ cái hoặc các ký hiệu đặc biệt khác.

7. Dụng cụ hiển thị 7-đoạn Anốt chung:

a. được biểu diễn bằng một Anốt đơn bên trong.

b. được biểu diễn bằng bảy đèn LED riêng lẻ.

c. được biểu diễn bằng một catốt đơn bên trong.

d. không có phương án nào đúng.

8. Bộ hợp kênh có khả năng:

a. nối một lối vào mạch với một lối ra trong một nhóm các lối ra.

b. nối đồng thời một lối vào mạch với một hoặc nhiều lối ra của một nhóm các lối ra.

c. nối một lối vào trong một nhóm các lối vào với một lối ra.

d. nối đồng thời một hoặc nhiều lối vào với một lối ra.

9. Bộ phân kênh có khả năng:

a. nối một lối vào mạch với một lối ra trong một nhóm các lối ra.

b. nối đồng thời một lối vào mạch với một hoặc nhiều lối ra trong một nhóm các lối ra.

c. nối một lối vào trong một nhóm các lối vào với một lối ra.

d. nối đồng thời một hoặc nhiều lối vào với một lối ra.

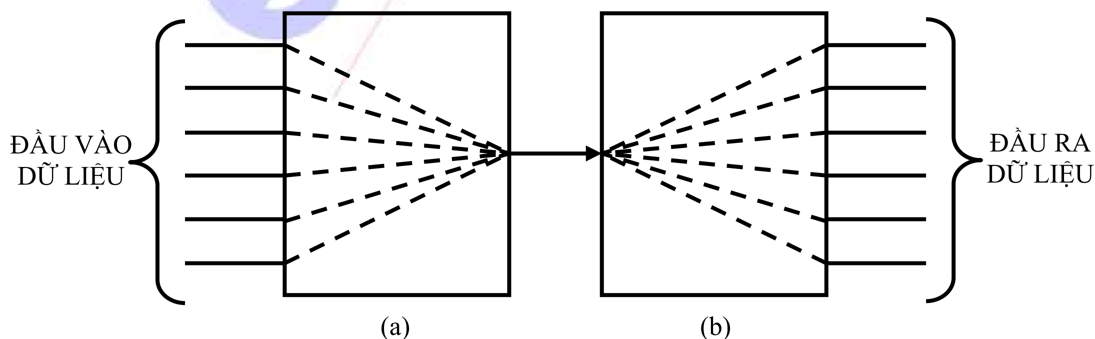
10. Mạch minh hoạ trong hình 4-29 là:

a. cặp giải mã (a)/ mã hóa (b).

b. cặp mã hoá (a)/ giải mã (b).

c. cặp hợp kênh (a)/phân kênh (b).

d. cặp phân kênh (a)/hợp kênh (b).



Hình 4-29.

11. IC trong hình 4-29(a) được gọi là:
    a. bộ hợp kênh 8 vào - 1 ra.
    b. bộ phân kênh 8 vào – 1 ra.
    c. bộ hợp kênh 1vào – 8 ra.
    d. bộ phân kênh 1vào – 8 ra.
12. IC trong hình 4-29(b) được gọi là:
    a. bộ hợp kênh 8 vào – 1 ra.
    b. bộ phân kênh 8 vào – 1 ra.
    c. bộ hợp kênh 1 vào – 8 ra.
    d. bộ phân kênh 1 vào – 8 ra.
13. Thuật ngữ *parity* (tính chẵn lẻ):
    a. dùng để chỉ kích thước đường dữ liệu của hệ thống.
    b. chỉ có thể dùng cho các hệ thống 8-bit.
    c. liên quan đến quá trình kiểm tra lỗi.
    d. dùng cho thanh ghi dịch.
14. Nếu bộ tạo bit chẵn lẻ nhận một bit kiểm tra parity chẵn, nó yêu cầu nhận:
    a. dữ liệu parity chẵn.
    b. dữ liệu parity lẻ.
    c. một trong hai trường hợp trên.
    d. Không phải hai trường hợp trên.
15. Khi ghép bộ cộng 2 số nhị phân 4 bit có thể :
    a. Cộng thành các số 8 bit.
    b. Cộng thành các số 4 bit.
    c. Tạo ra một tổng 8 bit.
    d. Tạo ra một số 8 bit khác.
16. Lối ra của từng tổng của bộ cộng có được là do thực hiện cộng :
    a. Tất cả 4 bit của từng số nhị phân.
    b. từng cặp bit một.
    c. Bit nhớ.
    d. 1 với bit trước đó.
17. Nếu lối ra A>B của bộ so sánh được kích hoạt, thì:

a. Giá trị của số A lớn hơn giá trị của số B.

b. Cả hai số ở lối vào đều có giá trị giống nhau.

c. Giá trị của số A nhỏ hơn giá trị của số B.

d. Giá trị của số B lớn hơn giá trị của số A.

18. Nếu lối ra A=B của bộ so sánh được kích hoạt, thì:

a. Giá trị của số A lớn hơn giá trị của số B.

b. Cả hai số ở lối vào đều có giá trị giống nhau.

c. Giá trị của số A nhỏ hơn giá trị của số B.

d. Giá trị của số B lớn hơn giá trị của số A.

19. Nếu lối ra A<B của bộ so sánh được kích hoạt, thì:

a. Giá trị của số A lớn hơn giá trị của số B.

b. Cả hai số ở lối vào đều có giá trị giống nhau.

c. Giá trị của số A nhỏ hơn giá trị của số B.

d. Giá trị của số B nhỏ hơn giá trị của số A.

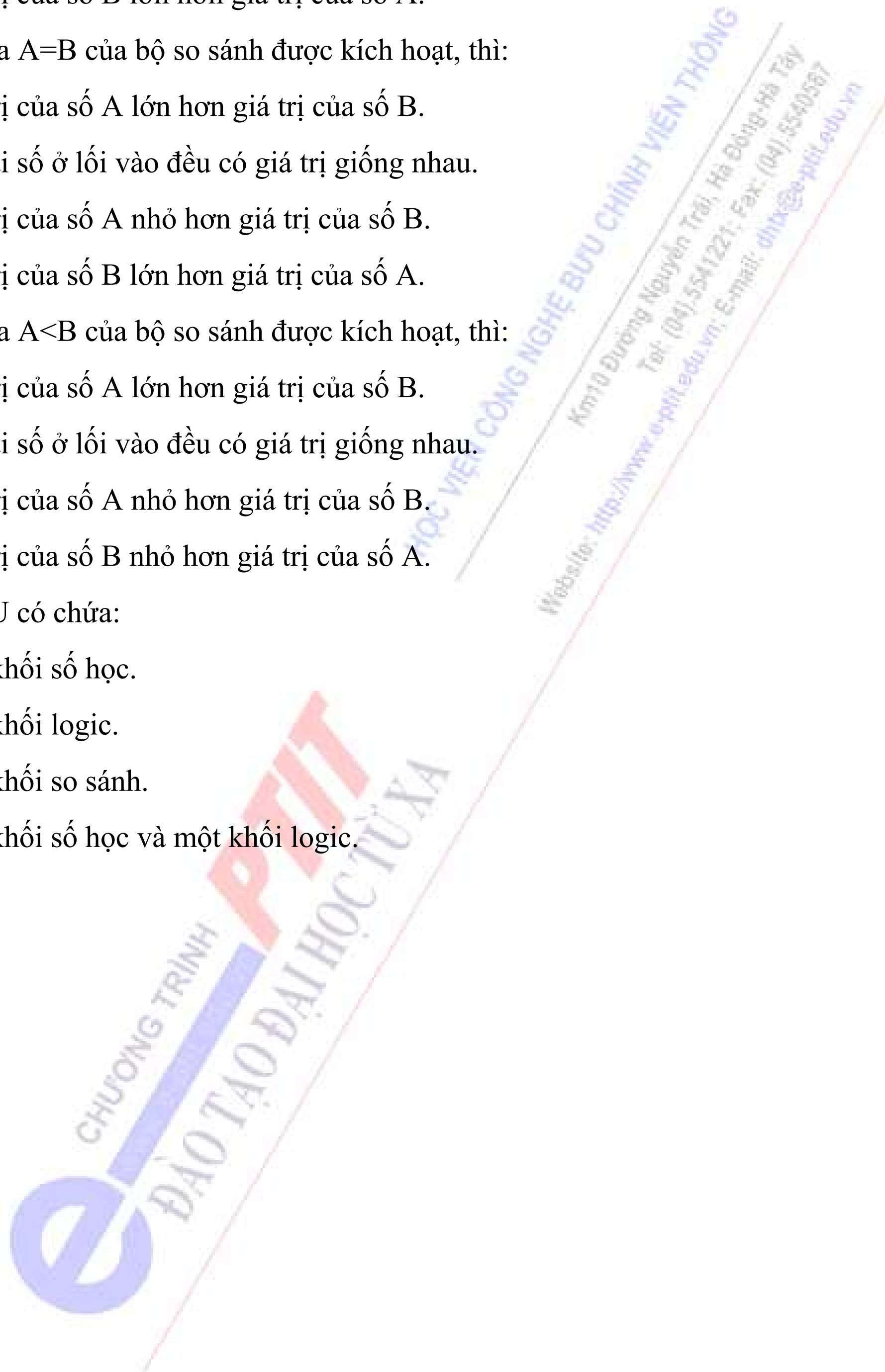
20. Một ALU có chứa:

a. Một khối số học.

b. Một khối logic.

c. Một khối so sánh.

d. Một khối số học và một khối logic.



CHƯƠNG 5: MẠCH LOGIC TUẦN TỰ

GIỚI THIỆU.

Chúng ta đã nghiên cứu về phép phân tích và thiết kế các mạch logic tổ hợp. Mặc dù rất qua trọng nhưng nó chỉ là một phần của các hệ thống kỹ thuật số. Một phần qua trọng của các hệ thống kỹ thuật số khác là phân tích và thiết kế mạch tuần tự. Tuy nhiên việc thiết kế các mạch tuần tự lại phụ thuộc vào việc thiết kế mạch tổ hợp đã được đề cập ở chương 4.

Có nhiều ứng dụng mà đầu ra số phải được tạo để phù hợp với tuần tự nhận được các tín hiệu vào. Yêu cầu này không thể được thoả mãn bằng việc sử dụng hệ thống logic tổ hợp.

Những ứng dụng này yêu cầu đầu ra không chỉ phụ thuộc vào các điều kiện đầu vào hiện có mà còn phụ thuộc vào lịch sử của các đầu vào. Lịch sử được cung cấp bằng cách phản hồi từ đầu ra về lại đầu vào.

Mạch logic tuần tự không những phụ thuộc vào trạng thái các lối vào và còn phụ thuộc vào trạng thái trong của nó. Mạch tuần tự được chia làm hai loại chính là mạch tuần tự không đồng bộ và mạch tuần tự đồng bộ.

Trong phần này chúng ta sẽ giới thiệu về các phần tử nhớ của mạch tuần tự. Cách phân tích và thiết kế mạch tuần tự đơn giản và phức tạp.

NỘI DUNG

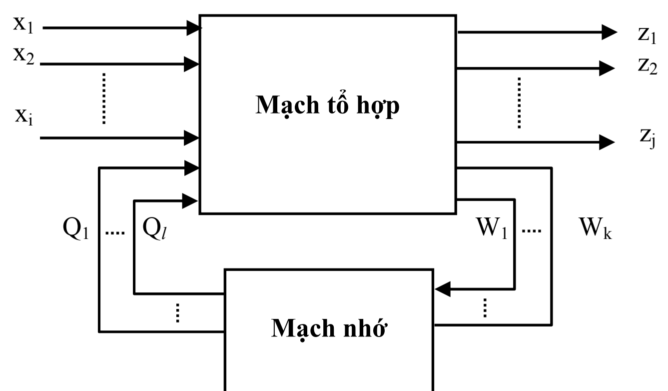
5.1. KHÁI NIỆM CHUNG VÀ MÔ HÌNH TOÁN HỌC

5.1.1. Khái niệm chung

Trong chương này, chúng ta sẽ nói đến hệ thống số được gọi là mạch logic tuần tự (hay còn gọi là mạch dãy - Sequential Circuit). Hoạt động của hệ này có tính chất kế tiếp nhau, tức là trạng thái hoạt động của mạch điện không những phụ thuộc trực tiếp lối vào mà còn phụ thuộc vào trạng thái bên trong trước đó của chính nó. Nói cách khác các hệ thống này làm việc theo nguyên tắc có nhớ.

5.1.2. Mô hình toán học

Mạch tuần tự là mạch bao gồm mạch logic tổ hợp và mạch nhớ. Mạch nhớ là các trigơ. Đối với mạch tuần tự, đáp ứng ra của hệ thống mạch điện không chỉ phụ thuộc trực tiếp vào tín hiệu vào (X) mà còn phụ thuộc vào trạng thái nội (Q) của nó. Có thể mô tả sơ đồ khối tổng quát của mạch tuần tự.



Hình 5-1. Sơ đồ khối của mạch tuần tự.

Ở đây: X - tập tín hiệu vào.

Q - tập trạng thái trong trước đó của mạch.

W - hàm kích.

Z - các hàm ra

Hoạt động của mạch tuần tự được mô tả bằng mối quan hệ toán học sau:

Z = f(Q, X)

Trong phương trình toán học của mạch tuần tự ta thấy có hai thông tin. Đó là thông tin về trạng thái tiếp theo của mạch tuần tự và thông tin về tín hiệu ra của mạch. Hai thông tin này cùng phụ thuộc đồng thời vào trạng thái bên trong trước đó của mạch (Q) và tín hiệu tác động vào (X) của nó. Ta có thể viết lại biểu thức trên như sau:

Z = f (Q(n), X).

Q (n +1) = f (Q(n), X)

Trong đó: Q(n +1): là trạng thái tiếp theo của mạch.

Q(n): là trạng thái bên trong trước đó. Để tiện cho việc nghiên cứu ta sẽ ký hiệu Q(n +1) là Q^k, Q(n) là Q.

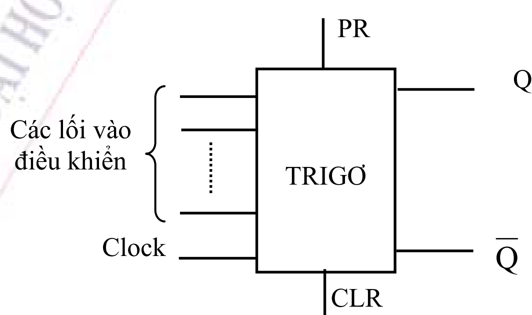
Để hiểu rõ hơn về mạch tuần tự ta đi xét các phần tử có trong mạch. Như ta đã biết mạch logic tổ hợp đã được xét ở chương 4. Bây giờ ta sẽ tìm hiểu về mạch nhớ, mà phần tử nhớ chính là các trigơ.

5.2. PHẦN TỬ NHỚ CỦA MẠCH TUẦN TỰ

5.2.1. Các loại Trigơ

Định nghĩa: Trigơ là phần tử có khả năng lưu trữ (nhớ) một trong hai trạng thái 0 và 1.

Trigơ có từ 1 đến một vài lối điều khiển, có hai lối ra luôn luôn ngược nhau là Q và $\overline{Q}$. Tuỳ từng loại trigơ có thể có thêm các lối vào lập (PRESET) và lối vào xoá (CLEAR). Ngoài ra, trigơ còn có lối vào đồng bộ (CLOCK). Hình 5-2 là sơ đồ khối tổng quát của trigơ.

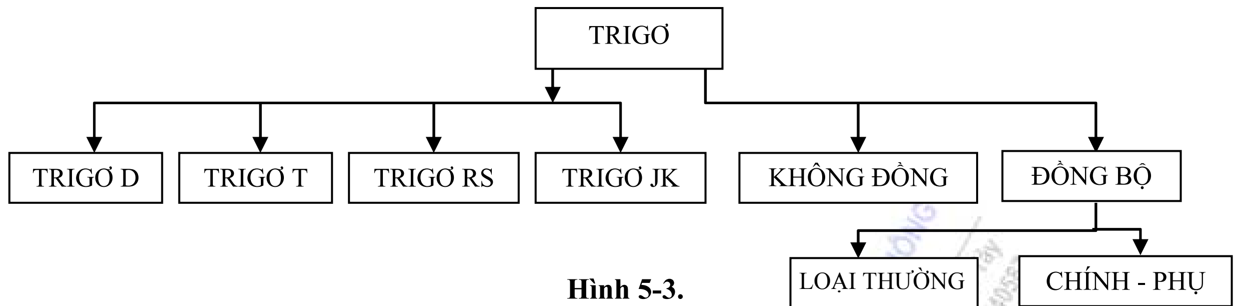


Hình 5-2. Sơ đồ tổng quát của một Trigơ

Phân loại:

- Theo chức năng làm việc của của các lối vào điều khiển: hiện nay thường sử dụng loại trigơ 1 lối vào như trigơ D, T; loại hai lối vào như trigơ RS, trigơ JK.
- Theo phương thức hoạt động thi ta có hai loại: trigơ đồng bộ và trigơ không đồng bộ. Trong loại trigơ đồng bộ lại được chia làm hai loại: trigơ thường và trigơ chính - phụ (Master- Slave).

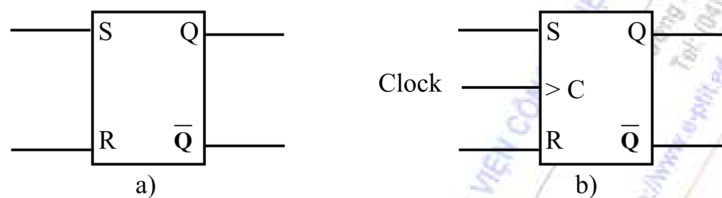
Sơ đồ khối của sự phân loại trigơ được cho ở hình 5-3.



Hình 5-3.

5 2.1.1. Trigơ RS

Trigơ RS là loại có hai lối vào điều khiển S, R. Chân S gọi là lối vào "lập" (SET) và R được gọi là lối vào "xoá" (RESET).



Hình 5-4. Sơ đồ ký hiệu của trigơ RS

Hình 5-4 là ký hiệu của trigơ RS trong các sơ đồ logic (hình a là sơ đồ của trigơ RS không đồng bộ, hình b là sơ đồ của trigơ RS đồng bộ). Hình 5-5 là sơ đồ nguyên lý của trigơ RS và RS đồng bộ. Trạng thái ở đầu ra của Q phụ thuộc vào các tín hiệu logic ở hai lối vào điều khiển S, R theo bảng trạng thái 5-1 và 5-2 :

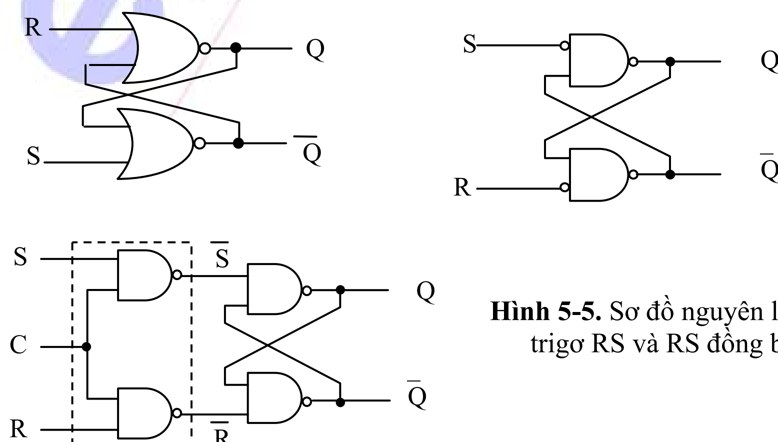
S	R	Q^k	Mod hoạt động
0	0	Q	Nhớ
0	1	0	Xoá
1	0	1	Lập
1	1	x	Cấm

Bảng 5-1. Bảng trạng thái của trigơ RS

C	S	R	Q^k	Mod hoạt động
0	x	x	Q	Nhớ
1	0	0	Q	Nhớ
1	0	1	0	Xoá
1	1	0	1	Lập
1	1	1	x	Cấm

Bảng 5-2. Bảng trạng thái của trigơ RS đồng bộ cổng NAND

Trong bảng, ký hiệu Q^k là giá trị ở lối ra Q ở thời điểm kế tiếp, Q là giá trị tại thời điểm hiện tại.



Hình 5-5. Sơ đồ nguyên lý của trigơ RS và RS đồng bộ

Ta thấy khi S = 1, R = 0 thì $Q^k = 1$; khi S = 0, R = 1 thì $Q^k = 0$. Đây chính là hai điều kiện điều khiển ở lối vào khiến cho lối ra của trigơ có thể lật trạng thái. S và R là các lối vào điều khiển. Trường hợp S = 0, R = 0 thì $Q^k = Q$, điều này có nghĩa là khi không có tín hiệu điều khiển thì trigơ vẫn giữ nguyên trạng thái vốn có của nó. Cuối cùng khi S = R = 1 thì lối ra Q^k và $\overline{Q^k}$ có giá trị bằng nhau (có thể là 1, có thể là 0) nên ta nói trạng thái của trigơ là không xác định hay gọi là trạng thái cấm. Vậy, ***không bao giờ được sử dụng trường hợp này.***

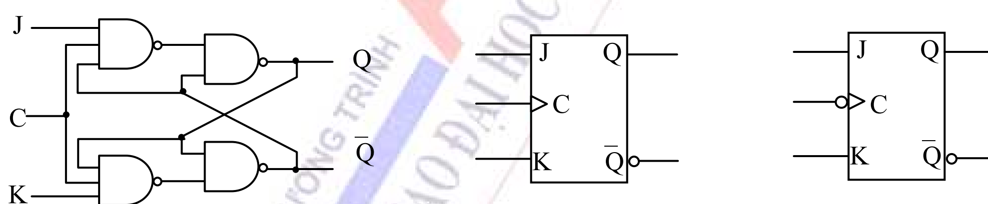
5 2.1.2. Trigơ JK

Trigơ JK là loại trigơ có hai lối vào điều khiển J, K. Trigơ này có ưu điểm hơn trigơ RS là không còn tồn tại tổ hợp cấm bằng các đường hồi tiếp từ Q về chân R và từ $\overline{Q}$ về S. Tuy nhiên, điểm đặc biệt là trigơ JK còn có thêm đầu vào đồng bộ C. Trigơ có thể lập hay xoá trong khoảng thời gian ứng với sườn âm hoặc sườn dương của xung đồng bộ C. Ta nói, trigơ RS thuộc loại ***đồng bộ.***

Sự hoạt động của trigơ JK được trình bày bằng bảng trạng thái 5-2

C	J	K	Q^k	Mod hoạt động
0	x	x	Q	Nhớ (đối với loại trigơ JK dùng cổng NAND)
1	x	x	Q	Nhớ (đối với loại trigơ JK dùng cổng NOR)
Ck	0	0	Q	Nhớ
Ck	0	1	0	Xoá
Ck	1	0	1	Lập
Ck	1	1	$\overline{Q}$	Thay đổi trạng thái theo mỗi xung nhịp

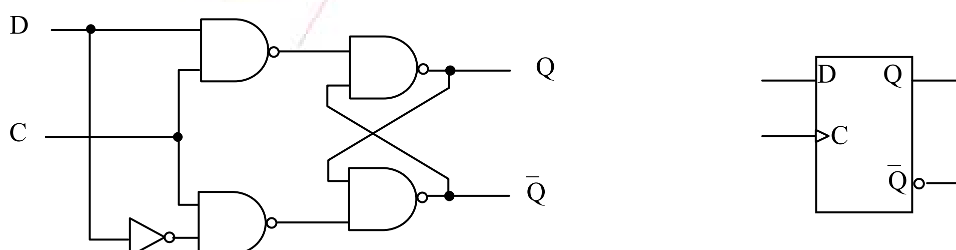
Bảng 5-3. Bảng trạng thái của trigơ JK đồng bộ



Hình 5-6. Sơ đồ nguyên lý và ký hiệu của trigơ JK đồng bộ

Sơ đồ nguyên lý và sơ đồ ký hiệu của trigơ JK được trình bày ở hình 5-6.

5 2.1.3. Trigơ D



Hình 5-7. Sơ đồ nguyên lý và ký hiệu của trigơ D đồng bộ

Trigơ D là loại trigơ có một lối vào điều khiển D. Tín hiệu ở lối vào điều khiển sẽ truyền tới lối ra Q ($Q^k = D$) mỗi khi xuất hiện xung nhịp C. Trigơ D thường được dùng làm bộ ghi dịch dữ liệu hay bộ chốt dữ liệu. Sơ đồ nguyên lý và sơ đồ ký hiệu của trigơ D được biểu diễn ở hình 5-7.

5 2.1.4. Trigơ T

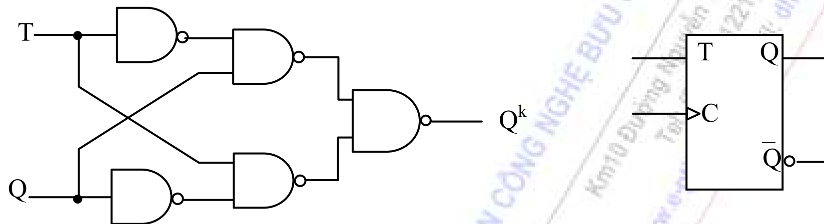
Trigơ T là loại trigơ có môt lối vào điều khiển T. Mỗi khi có xung tới lối vào T thì lối ra Q sẽ thay đổi trạng thái.

Bảng 5-3 là bảng trạng thái của trigơ T

Sơ đồ nguyên lý và ký hiệu của trigơ T được biểu diễn ở hình 5-8.

T	Q^K
0	Q
1	$\overline{Q}$

Bảng 5-4. Bảng trạng thái của trigơ T



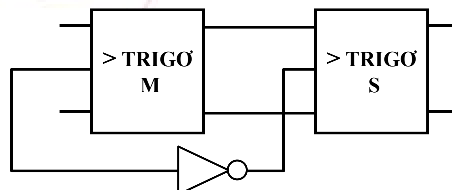
Hình 5-8. Sơ đồ nguyên lý và ký hiệu của trigơ T

- Nhận xét: Từ các bảng trạng thái của các trigơ trên ta thấy rằng: Các trigơ D và RS có thể làm việc được ở chế độ không đồng bộ vì mỗi tập tín hiệu vào điều khiển D, RS luôn luôn tồn tại ít nhất 1 trong 2 trạng thái ổn định. Trạng thái ổn định là trạng thái thoả mãn điều kiện $Q^k = Q$. Còn trigơ T và trigơ JK không thể làm việc được ở chế độ không đồng bộ vì mạch sẽ rơi vào trạng thái dao động nếu như tập tín hiệu vào là '11' đối với trigơ JK hoặc là '1' đối với trigơ T. Như vậy, trigơ D, trigơ RS có thể làm việc ở cả hai chế độ: đồng bộ và không đồng bộ còn trigơ T và trigơ JK chỉ có thể làm việc ở chế độ đồng bộ.

5 2.1.5. Các loại trigơ Chính- Phụ (MS-Master- Slave).

Do các loại trigơ đồng bộ trên đều hoạt động tại sườn dương hay sườn âm của xung nhịp nên khi làm việc ở tần số cao thì lối ra Q không đáp ứng kịp với sự thay đổi của xung nhịp, dẫn đến mạch hoạt động ở tình trạng không được tin cậy. Loại trigơ MS khắc phục được nhược điểm này. Lối ra của trigơ MS thay đổi tại sườn dương và sườn âm của xung nhịp, nên cấu trúc của nó gồm 2 trigơ giống nhau nhưng cực tính điều khiển của xung Clock thì ngược nhau để đảm bảo sao cho tại mỗi sườn của xung sẽ có một trigơ hoạt động. Về nguyên tắc hoạt động của loại trigơ MS (RS-MS, JK-MS, D-MS, T-MS) hoàn toàn giống như các loại trigơ thông thường (RS, JK, D, T).

Cấu trúc chung của một trigơ MS được minh hoạ ở hình 5-9.

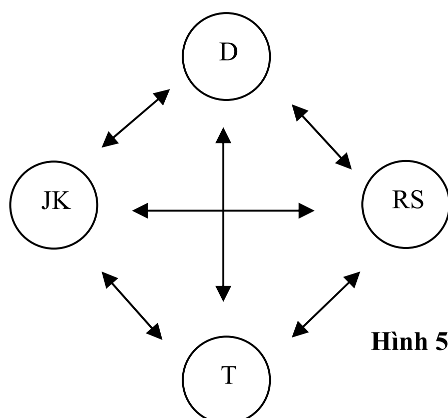


Hình 5-9. Cấu trúc của trigơ MS

5.2.2. Chuyển đổi giữa các loại trigơ.

Có 4 loại trigơ đã được giới thiệu là trigơ RS, JK, D và T. Trên thực tế có khi trigơ loại này lại được sử dụng như trigơ loại khác. Nội dung phần này là xây dựng các trigơ yêu cầu từ các trigơ cho trước.

Với 4 loại trigơ trên thì có 12 khả năng chuyển đổi sang nhau.



Hình 5-10. Các khả năng chuyển đổi giữa các loại trigơ.

5.2.2.1. Phương pháp chuyển đổi giữa các loại trigơ.

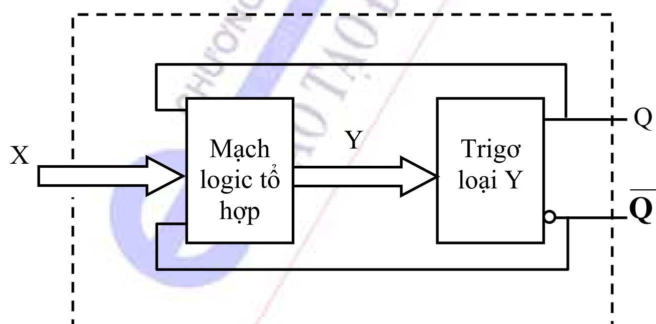
Một trong các phương pháp để xây dựng trigơ loại X từ loại Y cho trước được cho ở sơ đồ khối ở hình 5-11.

Các lối vào X là các lối vào của trigơ loại X cần thiết kế. Lối ra của mạch logic là các lối vào của trigơ Y cho trước. Như vậy, bài toán chuyển đổi từ trigơ loại Y sang trigơ loại X là xây dựng mạch tổ hợp có các đầu vào là X và Q; các lối ra là Y biểu diễn bởi hệ hàm:

Y = f (X, Q)

Để thực hiện chuyển đổi trigơ loại Y sang loại X cần thực hiện các bước sau:

- ♦ Xác định hệ hàm Y = f (X, Q) theo bảng hàm kích.
- ♦ Tối thiểu hoá các hàm này và xây dựng các sơ đồ.



Hình 5-11. Sơ đồ khối của trigơ loại X

Bảng hàm kích của các loại trigơ được cho ở bảng 5-5.

Q	Q^k	RS	JK	D	T
0	0	X0	0X	0	0
0	1	01	1X	1	1
1	0	10	X1	0	1
1	1	0X	X0	1	0

Bảng 5-5. Bảng hàm kích của các loại trigơ

Sau đây ta xét một số ví dụ xây dựng các trigơ từ các trigơ cho trước thường hay được sử dụng trong thực tế.

Ví dụ: Chuyển đổi từ trigơ RS sang trigơ JK.

Ta cần phải thiết kế mạch logic tổ hợp của các hàm logic:

$R = f_1 (Q, J, K)$

$S = f_2 (Q, J, K)$

Từ bảng hàm kích thích trên ta thu được bảng Karnaugh (bảng 5-6) cho S và R với các biến vào là Q, J, K.

Q \ JK	00	01	11	10
0	0	0	1	1
1	X	0	0	X

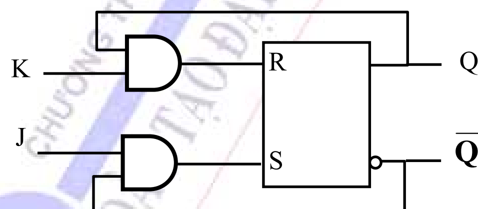
$S = J\overline{Q}$

Q \ JK	00	01	11	10
0	X	X	0	0
1	0	1	1	0

R = KQ

Bảng 5-6. Bảng tính S và R

Mạch thực hiện chuyển đổi được cho ở hình 5-12.



Hình 5-12. Trigơ JK xây dựng từ trigơ RS

Tương tự như vậy ta cũng có thể làm như vậy đối với các chuyển đổi khác.

5.3. PHƯƠNG PHÁP MÔ TẢ MẠCH TUẦN TỰ.

Thiết bị được thiết kế phải được mô tả bằng lời hay một số hình thức khác. Công việc đầu tiên của người thiết kế là phải phiên dịch các dữ kiện đó thành 1 hình thức mô tả hoạt động của thiết bị cần phải thiết kế một cách trung thực và duy nhất. Nói cách khác là phải hình thức hoá dưc liệu ban đầu.

Có hai cách hình thức hoá thường dùng đó là dùng bảng và đồ hình trạng thái.

5.3.1. Bảng

5.3.1.1. Bảng chuyển đổi trạng thái.

Bảng chuyển đổi trạng thái bao gồm các hàng và các cột, các hàng ghi các trạng thái trong, các cột ghi các giá trị của tín hiệu vào. Các ô ghi giá trị các trạng thái trong kế tiếp mà mạch sẽ chuyển đến ứng với các giá trị ở hàng và cột. Bảng chuyển đổi trạng thái được mô tả ở bảng 5-6.

Tín hiệu vào

V / S (Trạng thái trong)	V_1	V_2	....	V_n
S_1				
S_2				
:				
:				
S_n				

→ Trạng thái kế tiếp Q^k

Bảng 5-6. Bảng chuyển đổi trạng thái

5.3.1.2. Bảng tín hiệu ra.

Các hàng của bảng ghi các trạng thái trong, các cột ghi các tín hiệu vào. Các ô ghi giá trị của tín hiệu ra tương ứng. Bảng tín hiệu ra được mô tả ở bảng 5-7.

Tín hiệu vào

V / S (Trạng thái trong)	V_1	V_2	....	V_n
S_1				
S_2				
:				
:				
S_n				

→ Tín hiệu ra - R

Bảng 5-7. Bảng tín hiệu ra

Có thể gộp hai bảng chuyển đổi trạng thái và bảng tín hiệu ra thành một bảng chung gọi là bảng chuyển đổi trạng thái / ra. Lúc đó trên các ô ghi các giá trị của trạng thái kế tiếp và tín hiệu ra (S^k / R) tương ứng với trạng thái hiện tại và tín hiệu vào.

Bảng chuyển đổi trạng thái và tín hiệu ra được mô tả ở bảng 5-8.

Tín hiệu vào

V / S (Trạng thái trong)	V_1	V_2	....	V_n
S_1				
S_2				
:				
:				
S_n				

→ Trạng thái kế tiếp S^k và Tín hiệu ra - R

Bảng 5-8. Bảng chuyển đổi trạng thái và tín hiệu ra

5.3.2. Đồ hình trạng thái.

Đồ hình trạng thái là hình vẽ phản ánh quy luật chuyển đổi trạng thái và tình trạng các giá trị ở lối vào và lối ra tương ứng của mạch tuần tự.

Đồ hình trạng thái là một đồ hình có hướng gồm hai tập:

M - Tập các đỉnh và K - Tập các cung có hướng.

a). Đối với mô hình Mealy thực hiện ánh xạ.

Tập các trạng thái trong là tập các đỉnh M; Tập các tín hiệu vào / ra là tập các cung K.

Trên cung có hướng đi từ trạng thái trong S_i đến trạng thái trong S_j ghi tín hiệu vào/ra tương ứng.

b). Đối với mô hình Moore.

Vì tín hiệu ra chỉ phụ thuộc vào trạng thái trong của mạch mà không phụ thuộc vào tín hiệu vào cho nên thực hiện ánh xạ:

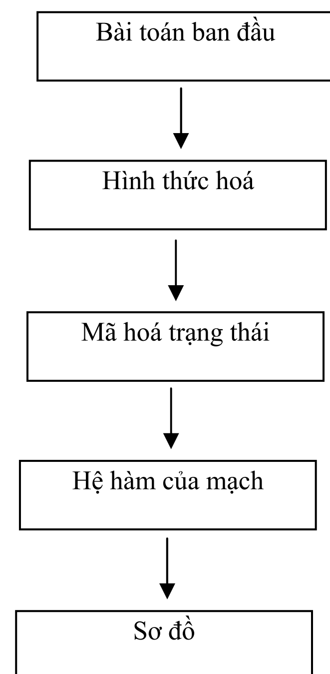
Tập các trạng thái trong, tín hiệu ra là tập các đỉnh M.

Tập các tín hiệu vào là tập các cung K.

5.4. CÁC BƯỚC THIẾT KẾ MẠCH TUẦN TỰ.

Quá trình thiết kế mạch tuần tự được mô tả theo lưu đồ sau

- ***Bài toán ban đầu:*** Nhiệm vụ thiết kế được mô tả bằng ngôn ngữ hoặc bằng lưu đồ thuật toán.
- ***Hình thức hoá:*** Từ các dữ kiện đề bài cho mà ta mô tả hoạt động của mạch bằng cách hình thức hoá dữ kiện ban đầu ở dạng bảng trạng thái, bảng ra hay đồ hình trạng thái. Sau đó rút gọn các trạng thái của mạch để có được số trạng thái trong ít nhất.
- ***Mã hoá trạng thái:*** Mã hoá tín hiệu vào ra, trạng thái trong để nhận được mã nhị phân (hoặc có thể là các loại mã khác) có tập tín hiệu vào là X, tập tín hiệu ra là Y, tập các trạng thái trong là Q.
- ***Hệ hàm của mạch:*** Xác định hệ phương trình logic của mạch và tối thiểu hoá các phương trình này. Nếu mạch tuần tự khi thiết kế cần dùng các trigơ và mạch tổ hợp thì tuỳ theo yêu cầu mà ta viết hệ phương trình cho các lối vào kích cho từng loại trigơ đó.
- ***Xây dựng sơ đồ:*** Từ hệ phương trình của mạch đã viết được ta xây dựng mạch điện thực hiện.



Hình 5-13. Các bước thiết kế mạch tuần tự

5.4.1. Thiết kế mạch tuần tự từ đồ hình trạng thái.

Giả thiết: Cho đồ hình trạng thái của mạch có tập tín hiệu vào V, tập tín hiệu ra R, tập trạng thái trong S (chưa mã hoá nhị phân).

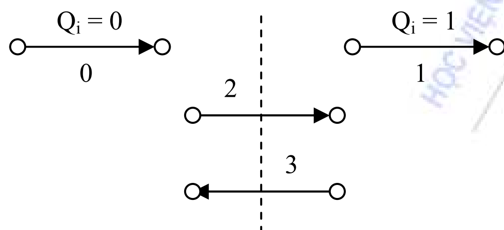
Xác định: Hệ phương trình nhị phân của mạch (đã tối thiểu hoá). Trên cơ sở đó vẽ mạch điện.

5.4.1.1. Các bước thiết kế

- Mã hoá tín hiệu vào V, tín hiệu ra R, trạng thái trong S để chuyển thành mạch dạng nhị phân có các tập tín hiệu vào X, tín hiệu ra Y, trạng thái trong Q.
- Xác định hệ phương trình tín hiệu ra: $Y_i = f_i (X, Q)$. Phương trình này được xác định trên các cung với mô hình kiểu Mealy, trên các đỉnh với mô hình kiểu Moore. Tối thiểu các hàm này.
- Xác định hệ phương trình hàm kích cho các trigơ và tối thiểu hoá nó.

Sau đây giới thiệu thuật toán xác định phương trình lối vào kích cho các trigơ từ đồ hình trạng thái.

Đối với trigơ Q_i bất kỳ sự thay đổi trạng thái từ Q_i đến Q^k_i chỉ có thể có 4 khả năng như hình 5-16.



Hình 5-14. Các cung biểu diễn sự thay đổi trạng thái từ Q_i đến Q^k_i của trigơ Q_i

Trong đó các cung biểu diễn sự thay đổi từ Q_i đến Q^k_i được ký hiệu như sau: $0 \rightarrow 0$ là (0), $1 \rightarrow 1$ (là 1), $0 \rightarrow 1$ là (2), $1 \rightarrow 0$ là (3).

Từ quy ước có thuật toán sau:

a.Thuật toán xác định phương trình lối vào kích cho trigơ Q_i loại D.

Phương trình đặc trưng của trigơ D : $Q^k_i = D_i$. Từ đó ta rút ra

$D_i = Q^k_i$ = tuyển tất cả các cung đi tới đỉnh có $Q_i = 1$.

$= \sum$ các cung loại (2), kể cả khuyên tại đỉnh đó tức là cung loại 1

$= \sum$ (1) và (2)

Tối thiểu hoá hàm D_i vừa tìm được rút ra phương trình lối vào kích cho trigơ loại D.

b.Thuật toán xác định phương trình lối vào kích cho trigơ T

Phương trình đặc trưng của trigơ T: $Q^k_i = T_i \oplus Q_i \Rightarrow T_i = Q_i \oplus Q^k_i = Q'_i$

Trong đó Q'_I bằng 1 khi Q_i thay đổi trạng thái từ $0 \Rightarrow 1$ hoặc từ $1 \Rightarrow 0$, ta làm như sau:

- Điền sự thay đổi giá trị của Q_i vào các cung.

- $T_i = Q'_I = \sum$ các cung có Q_i thay đổi (cung loại 2, loại 3) $= \sum$ (2) và (3).

Tối thiểu hoá hàm T_i vừa tìm được rút ra phương trình kích cho trigơ T.

c.Thuật toán xác định phương trình lối vào kích cho trigơ JK

Phương trình đặc trưng của trigơ JK: $Q_i^k = J\,\overline{Q_i} + \overline{K}\,Q_i$

Xác định:

T_{on} = $\sum$ các cung mà Q_i được bật (Q_i thay đổi từ 0 $\Rightarrow$ 1 - cung loại 2) = $\sum$ (2). Đưa phương trình của T_{on} về dạng:

$$T_{on} = (\,T^*\,)\,\overline{Q_i} \;\Rightarrow \text{rút ra } J = T^*.$$

T_{off} = $\sum$ các cung mà Q_i được tắt (Q_i thay đổi từ 1 $\Rightarrow$ 0 - cung loại 3) = $\sum$ (3). Đưa phương trình của T_{off} về dạng:

$$T_{off} = (\,T^{**}\,)\,\overline{Q_i} \;\Rightarrow \text{rút ra } K = T^{**}.$$

d.Thuật toán xác định phương trình lối vào kích cho trigơ RS

Phương trình lối vào S của trigơ RS được xác định như sau:

$S = T_{on}$ + [Các cung loại (1)]

$R = T_{off}$ + [Các cung loại (0)]

Các cung loại (1), các cung loại (0) để trong dấu [ ] ở biểu thức của S, R được lấy giá trị không xác định. Những giá trị này và những trạng thái không được sử dụng sẽ được dùng để tối thiểu hoá sao cho biểu thức nhận được là tối giản nhất.

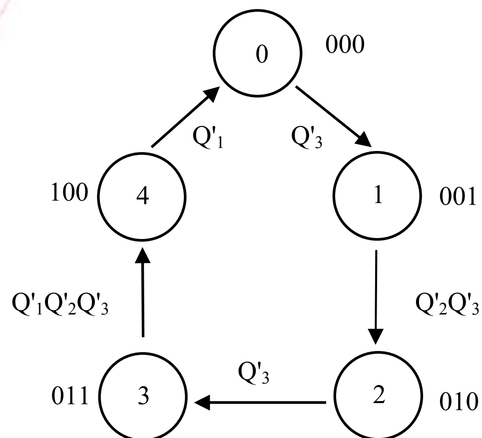
5.4.1.2. Ví dụ

Để minh hoạ,xét ví dụ sau: Thiết kế bộ đếm đồng bộ có $M_đ$ = 5 với đồ hình trạng thái và mã hoá trạng thái như ở hình 5-17 , dùng

Q_2Q_3 \ Q_1	00	01	11	10
0	0	1	3	2
1	4	x	x	x

b). Bảng mã hoá trạng thái

Hình 5-15.



a). Đồ hình trạng thái

a) Trigơ D và các mạch AND.

b) Trigơ T và các mạch AND.

c) Trigơ JK và các mạch AND.

d) Trigơ RS và các mạch AND.

Bộ đếm M =5 nên có 5 trạng thái 0, 1, 2, 3, 4. Để đơn giản, trên đồ hình ta không ghi các tín hiệu vào đếm và tín hiệu ra. Tín hiệu ra của bộ đếm chỉ xuất hiện khi bộ đếm đang ở trạng thái 4 và có tín hiệu vào đếm, lúc đó bộ đếm quay trở về trạng thái ban đầu 0 và cho ra tín hiệu ra.

Mạch có 5 trạng thái và do vậy được mã hoá ít nhất bằng 3 biến nhị phân tương ứng với 3 trigơ: Q_1, Q_2 Q_3 như trên bảng mã hoá trạng thái hình 5-17b. Điền mã tương ứng vào các trạng thái trên đồ hình 5-17a.

Từ đó ta viết được phương trình cho tín hiệu ra Y: $Y = Q_1\overline{Q_2}\,\overline{Q_3}\ .\ X_đ$.

Sử dụng các trạng thái tuỳ chọn để tối thiểu hoá, từ đó ta nhận được kết quả

$$Y = Q_1 X_đ$$

Bây giờ ta xác định các phương trình kích cho các trigơ :

a) Trigơ D.

Nhìn vào đồ hình trạng thái ta thấy: $Q_1 = 1$ tại đỉnh (4), $Q_2 = 1$ tại đỉnh (2), (3), $Q_2 = 1$ tại đỉnh (1), (3).

$D_1 = \sum$ Các cung đi đến đỉnh (4) = (3) = $\overline{Q_1}Q_2Q_3$.

$D_2 = \sum$ Các cung đi đến đỉnh (2), (3) = (1) + (2) = $\overline{Q_1}\,\overline{Q_2}Q_3 + \overline{Q_1}Q_2\overline{Q_3}$.

$D_3 = \sum$ Các cung đi đến đỉnh (1), (3) = (0) + (2) = $\overline{Q_1}\,\overline{Q_2}\,\overline{Q_3} + \overline{Q_1}Q_2\overline{Q_3}$.

Q_2Q_3 \ Q_1	00	01	11	10
0			1	
1		x	x	x

$D_1 = Q_2Q_3$

Q_2Q_3 \ Q_1	00	01	11	10
0		1		1
1		x	x	x

$D_2 = \overline{Q_2}Q_3 + Q_2\overline{Q_3} = Q_2 \oplus Q_3$

Q_2Q_3 \ Q_1	00	01	11	10
0	1			1
1		x	x	x

$D_3 = \overline{Q_1}\,\overline{Q_3}$

Bảng 5-9. Bảng tìm hàm kích

Dùng bảng Karnaugh 5-9 ta thu được kết quả

$$D_1 = Q_2Q_3$$

$$D_2 = \overline{Q_2}Q_3 + Q_2\overline{Q_3} = Q_2 \oplus Q_3$$

$D_3 = \overline{Q_1}\,\overline{Q_3}$

b) Xác định phương trình kích cho Trigơ T.

Điền sự thay đổi giá trị của Q_i (Q_i) vào các cung. Khi mạch đếm từ trạng thái (0) ⇒ (1) (nghĩa là từ 000 ⇒ 001) thì Q_3 thay đổi từ 0 ⇒ 1 nên ta ghi Q_3 lên cung đó. Khi mạch chuyển từ trạng thái (1) ⇒ (2) (tương ứng từ 001 ⇒ 010): Q_1 không thay đổi trạng thái (= 0), Q_2 thay đổi từ 0 ⇒ 1 và Q_3 thay đổi từ 1 ⇒ 0, nên ta ghi Q_2Q_3 lên cung từ (1) ⇒ (2). Tương tự như vậy ta có:

$T_1 = Q_1 = \sum$ các cung có Q_1 thay đổi = (3) + (4) = $\overline{Q_1}\,Q_2\,Q_3 + Q_1\,\overline{Q_2}\,\overline{Q_3}$

$T_2 = Q_2 = \sum$ các cung có Q_2 thay đổi = (1) + (3) = $\overline{Q_1}\,\overline{Q_2}\,Q_3 + \overline{Q_1}\,Q_2Q_3$

$T_3 = Q_3 = \sum$ các cung có Q_3 thay đổi = (0) + (1) + (2) + (3) = $\overline{Q_1}\,\overline{Q_2}\,\overline{Q_3} + \overline{Q_1}\,\overline{Q_2}\,Q_3 + \overline{Q_1}\,Q_2\,\overline{Q_3} + \overline{Q_1}\,Q_2Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0			1	
1	1	x	x	x

$T_1 = Q_1 + Q_2Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0		1	1	
1		x	x	x

$T_2 = Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0	1	1	1	1
1		x	x	x

Bảng 5-10.

$T_3 = \overline{Q_1}$

Lập bảng Karnaugh 5-10 cho các hàm trên ta thu được kết quả:

$T_1 = Q_1 + Q_2Q_3$

$T_2 = Q_3$

$T_3 = \overline{Q_1}$

c) Xác định phương trình kích cho Trigơ JK.

Chú ý khi viết các biểu thức T_{on}, T_{off} của trigơ thứ I ta cần phải đơn giản các biểu thức đó và đưa về dạng:

$T_{on} = (\,T^*\,)\ \overline{Q_i}$ ⇒ rút ra $J_i = T^*$.

$T_{off} = (\,T^{**}\,)\ \overline{Q_i}$ ⇒ rút ra $K_i = T^{**}$.

Viết các biểu thức T_{on}, T_{off} cho các trigơ và từ đó xác định phương trình kích cho các trigơ như sau:

T_{on1} = Σ Các cung mà Q_1 được bật (Chuyển từ 0 ⇒ 1) = (3) = $\overline{Q_1}Q_2Q_3$

T_{off1} = Σ Các cung mà Q_1 tắt (Chuyển từ 1 ⇒ 0) = (4) = $Q_1\overline{Q_2}\,\overline{Q_3}$

T_{on2} = Σ Các cung mà Q_2 được bật (Chuyển từ 0 ⇒ 1) = (1) = $\overline{Q_1}\,\overline{Q_2}Q_3$

T_{off2} = Σ Các cung mà Q_2 tắt (Chuyển từ 1 ⇒ 0) = (3) = $\overline{Q_1}Q_2Q_3$

T_{on3} = Σ Các cung mà Q_3 được bật (Chuyển từ 0 ⇒ 1) = (0) + (2) = $\overline{Q_1}\,\overline{Q_3}$

T_{off3} = Σ Các cung mà Q_3 tắt (Chuyển từ 1 ⇒ 0) = (1) + (3) = $\overline{Q_1}Q_3$

Biểu diễn các hàm này trên bảng Karnaugh, sử dụng các trạng thái tuỳ chọn để tối thiểu hoá . Các trạng thái tuỳ chọn bao gồm 3 số không nằm trong phạm vi đếm 5, 6, 7. Ngoài ra còn một số trạng thái khác tuỳ vào từng bảng. Ví dụ, đối với bảng tính J_1 giá trị tuỳ chọn ngoài 3 số trên còn thêm ô có giá trị Q_1 = 1, bảng tính K_1 có thêm các ô có giá trị Q_1 = 0, tương tự như vậy với các bảng còn lại.

Q_1 \ Q_2Q_3	00	01	11	10
0			1	
1	x	x	x	x

$J_1 = Q_2Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0	x	x	x	x
1	1	x	x	x

$K_1 = 1$

Q_1 \ Q_2Q_3	00	01	11	10
0		1	x	
1		x	x	x

$J_2 = Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0	x	x	1	
1	x	x	x	x

$K_2 = Q_3$

Q_1 \ Q_2Q_3	00	01	11	10
0	1	x	x	1
1		x	x	x

$J_3 = \overline{Q_1}$

Q_1 \ Q_2Q_3	00	01	11	10
0	x	1	1	x
1	x	x	x	x

$K_3 = 1$

Bảng 5-11. Bảng tìm hàm kích

Ta thu được kết quả từ bảng 5-11 như sau:

$$J_1 = Q_2Q_3; \quad K_1 = 1$$

$$J_2 = Q_3; \quad K_2 = Q_3$$

$$J_3 = \overline{Q_1}; \quad K_3 = 1$$

d) Xác định phương trình kích cho Trigơ RS.

$S_1 = T_{on1}$ + [Các cung loại (1)] = (3) + [ϕ]

$R_1 = T_{off1}$ + [Các cung loại (0)] = (4) + [(0), (1), (2)]

$S_2 = T_{on2}$ + [Các cung loại (1)] = (1) + [(2)]

$R_2 = T_{off2}$ + [Các cung loại (0)] = (3) + [(0), (4)]

$S_3 = T_{on3}$ + [Các cung loại (1)] = (0) + (2) + [ϕ]

$R_3 = T_{off3}$ + [Các cung loại (0)] = (1) + (3) + [(4)]

Biểu diễn các hàm này trên bảng Karnaugh và tối thiểu hoá chúng.

Q_2Q_3 \ Q_1	00	01	11	10
0			1	
1		x	x	x

$S_1 = Q_2Q_3$

Q_2Q_3 \ Q_1	00	01	11	10
0	x	x		x
1	1	x	x	x

$R_1 = Q_1$ hoặc $R_1 = \overline{Q_2}$ hoặc $R_1 = \overline{Q_3}$

Q_2Q_3 \ Q_1	00	01	11	10
0		1		x
1		x	x	x

$S_2 = \overline{Q_2}\,Q_3$

Q_2Q_3 \ Q_1	00	01	11	10
0	x		1	
1	x	x	x	x

$R_2 = Q_2\,Q_3$

Q_2Q_3 \ Q_1	00	01	11	10
0	1			1
1		x	x	x

$S_3 = \overline{Q_1}\,\overline{Q_3}$

Q_2Q_3 \ Q_1	00	01	11	10
0		1	1	
1	x	x	x	x

$R_3 = \overline{Q_2}\,Q_3$

Bảng 5-12. Bảng tìm hàm kích

Sau khi rút gọn từ bảng 5-12 ta thu được kết quả sau:

$S_1 = Q_2Q_3$; $R_1 = Q_1$ hoặc $R_1 = \overline{Q_2}$ hoặc $R_1 = \overline{Q_3}$

$S_2 = \overline{Q_2}\,Q_3$; $R_2 = Q_2\,Q_3$

$S_3 = \overline{Q_1}\,\overline{Q_3}$; $R_3 = \overline{Q_2}\,Q_3$

5.4.2. Thiết kế mạch tuần tự từ bảng.

Giả thiết: Cho bảng chuyển đổi trạng thái, bảng ra của mạch (chưa mã hoá nhị phân).

Xác định: Hệ phương trình nhị phân của mạch vào gồm hệ hàm ra, hệ hàm kích cho các trigơ. Trên cơ sở đó vẽ sơ đồ mạch.

Các bước thực hiện:

- Mã hoá tín hiệu vào V, tín hiệu ra R, trạng thái trong của mạch S để chuyển mạch ban đầu thành mạch nhị phân có tập tín hiệu vào X, tập tín hiệu ra Y, tập trạng thái trong Q.
- Lập bảng chuyển đổi trạng thái, bảng ra của mạch nhị phân ứng với sự mã hoá trên.
- Dựa vào bảng các lối vào kích của các trigơ xác định các lối vào kích cho các trigơ ứng với sự chuyển đổi trong bảng trạng thái.
- Viết phương trình lối vào kích cho từng Q_i của trigơ và các hàm ra rồi tối thiểu các hàm này. Trên cơ sở đó xây dựng mạch điện.

Ví dụ: Thiết kế bộ đếm có $K_đ = 5$, đồ hình trạng thái cho ở hình 5- 15a. Từ đó lập bảng chuyển đổi trạng thái như hình 5- 16a, mã hoá trạng thái như hình 5-16b. Dựa vào hai bảng này và căn cứ vào bảng hàm kích thích cho trigơ ở hình 5- 16c ta lập được bảng như hình 5-16d. Từ đó xác định được các phương trình các lối vào kích cho các loại trigơ. Bảng Karnaugh và kết quả tối giản giống như ở mục 5.4.1.2.

S	S^k
0	1
1	2
2	3
3	4
4	0

a) Bảng chuyển đổi trạng thái

Q_2Q_3 / Q_1	00	01	11	10
0	0	1	3	2
1	4	x	x	x

b) Bảng mã hoá trạng thái

Q	Q^k	D	T	R S	J K
0	0	0	0	X 0	0 X
0	1	1	1	0 1	1 X
1	0	0	0	1 0	X 1
1	1	1	1	0 X	X 0

c) Bảng hàm kích cho các trigơ

Q_1	Q_2	Q_3	$Q^k{}_1$	$Q^k{}_2$	$Q^k{}_3$	D_1	D_2	D_3	T_1	T_2	T_3	R_1S_1	R_2S_2	R_2S_3	J_1K_1	J_2K_2	J_3K_3
0	0	0	0	0	1	0	0	1	0	0	1	X 0	X 0	0 1	0 X	0 X	1 X
0	0	1	0	1	0	0	1	0	0	1	1	X 0	0 1	1 0	0 X	1 X	X 1
0	1	0	0	1	1	0	1	1	0	0	1	X 0	0 X	0 1	0 X	X 0	1 X
0	1	1	1	0	0	1	0	0	1	1	1	0 1	1 0	1 0	1 X	X 1	X 1
1	0	0	0	0	0	0	0	0	1	0	0	1 0	X 0	X 0	X 1	0 X	0 X
1	0	1	X	X	X	X	X	X	X	X	X	X X	X X	X X	X X	X X	X X
1	1	0	X	X	X	X	X	X	X	X	X	X X	X X	X X	X X	X X	X X
1	1	1	X	X	X	X	X	X	X	X	X	X X	X X	X X	X X	X X	X X

d) Bảng trạng thái nhị phân và đầu vào kích cho các loại trigơ

Hình 5-16. (a), (b), (c), (d) : Các bước thiết kế mạch tuần tự

5.5 MẠCH TUẦN TỰ ĐỒNG BỘ

Phần này trình bày phương pháp cơ bản để phân tích và thiết kế mạch tuần tự đồng bộ. Mạch tuần tự đồng bộ là một mạch số bao gồm các mạch tổ hợp và các phần tử nhớ (trigơ), hoạt động của mạch được đồng bộ bởi xung nhịp C. Trên thực tế để giảm nhỏ công suất tiêu thụ, thời gian trễ và để cho các mạch thực hiện đơn giản, người ta thường thiết kế sơ đồ sử dụng các trigơ JK và các mạch NAND.

Để nắm vững các vấn đề thiết kế mạch tuần tự đồng bộ, trước hết ta sẽ đi phân tích mạch tuần tự.

5.5.1. Phân tích mạch tuần tự đồng bộ.

5.5.1. 1. Các bước phân tích mạch tuần tự đồng bộ.

Bài toán phân tích là bài toán xác định chức năng của một mạch cho trước. Khi tiến hành phân tích cần tuân theo các bước sau:

- ***Sơ đồ mạch***: Từ sơ đồ cho trước cần xác định chức năng từng phần tử cơ bản của sơ đồ, mối quan hệ giữa các phần tử đó.

- ***Xác định các đầu vào và ra, số trạng thái trong của mạch:*** Coi mạch như một hộp đen cần phải xác định các đầu vào và ra của mạch, đặc điểm của các đầu vào, đầu ra. Để xác định được số trạng thái trong của mạch cần phải xác định xem mạch được xây dựng từ bao nhiêu phần tử nhớ (trigơ JK) từ đó xác định được số trạng thái trong có thể có của mạch.

Gọi số trigơ là *n* thì số trạng thái có thể có của mạch là 2^n.

- Xác định phương trình hàm ra, phương trình hàm kích của các trigơ.

- Lập bảng trạng thái, bảng ra nhị phân là bảng biểu diễn mối quan hệ trạng thái kế tiếp, tín hiệu ra nhị phân với trạng thái hiện tại và các tín hiệu vào tương ứng .

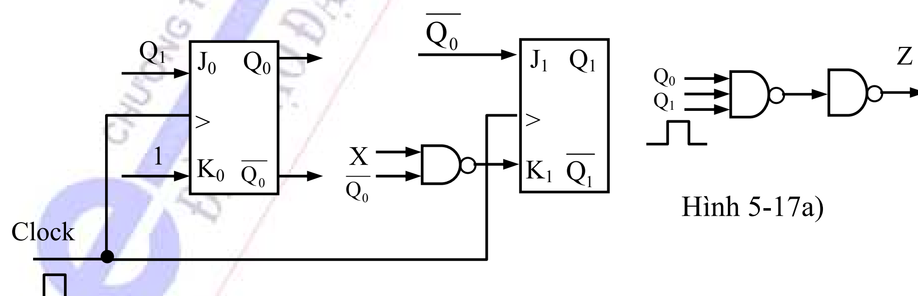
Dựa vào phương trình đặc trưng của trigơ xác định được trạng thái kế tiếp và tín hiệu ra tương ứng với tín hiệu vào và trạng thái hiện tại của mạch.

- ***Đồ hình trạng thái:*** Từ bảng trạng thái xây dựng đồ hình trạng thái và tín hiệu ra của mạch.

- ***Chức năng của mạch:*** Dựa vào đồ hình trạng thái xác định được chức năng của mạch

5.5.1.2. Ví dụ.

Phân tích mạch tuần tự đồng bộ có sơ đồ được biểu diễn như hình 5- 17a.



Hình 5-17a)

Bước 1. Sơ đồ trên có hai đầu vào là tín hiệu X và xung nhịp Clock. Có một tín hiệu Z ra, mạch sử dụng hai phần tử nhớ là hai trigơ JK (Q_0 và Q_1).

Bước 2: Xác định đầu vào, đầu ra và số trạng thái trong của mạch.

Mạch này có thể được biểu diễn bằng một "hộp đen" có hai đầu vào và một đầu ra. Do mạch được cấu tạo bằng hai trigơ nên số trạng thái có thể có của mạch là 4. Cụ thể là:

Q_1Q_0 = 00. 01, 10 và 11.

Bước 3: Xác định phương trình hàm ra và hàm kích cho trigơ.

Từ sơ đồ trên ta tìm được:

+ Phương trình hàm ra:

$$Z = C\,Q_1 Q_0$$

+ Phương trình hàm kích

$$J_0 = Q_1;\;\; K_0 = 1$$

$$J_1 = \overline{Q_0}\,;\; K_1 = \overline{X\,\overline{Q_0}} = \overline{X} + Q_0$$

Bước 4. Bảng chuyển đổi trạng thái.

Phương trình đặc trưng của trigơ JK là $Q^k = J\overline{Q} + \overline{K}Q$

Phương trình chuyển đổi trạng thái:

$$Q_0^k = J_0\overline{Q_0} + \overline{K_0}Q_0 = Q_1\overline{Q_0}$$

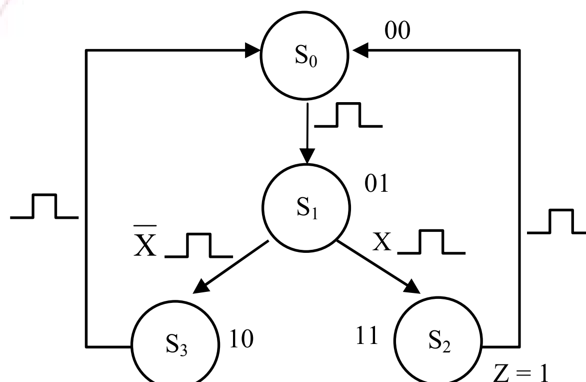
$$Q_1^k = J_1\overline{Q_1} + \overline{K_1}Q_1 = \overline{Q_0}\,\overline{Q_1} + \overline{\overline{X} + Q_0}\,Q_1 = \overline{Q_0}\,\overline{Q_1} + X\overline{Q_0}Q_1$$

Từ các phương trình trên ta lập được bảng chuyển đổi trạng thái

	Trạng thái hiện tại	**Trạng thái kế tiếp**		**Tín hiệu ra**	
		X = 0	**X = 1**	**X = 0**	**X = 1**
	Q_0Q_1	**Q_0Q_1**	**Q_0Q_1**	**Z**	**Z**
S_0	**00**	**01**	**01**	**0**	**0**
S_1	**01**	**10**	**11**	**0**	**0**
S_2	**11**	**00**	**00**	**1**	**1**
S_3	**10**	**00**	**00**	**0**	**0**

Hình 5-17 b). Bảng chuyển đổi trạng thái

Bước 5: Đồ hình trạng thái. Từ bảng chuyển đổi trạng thái trên ta xây dựng được đồ hình trạng thái như hình 5-17 c) (mô hình Mealy). Đồ hình gồm 4 trạng thái trong S_0, S_1, S_2, S_3. Các trigơ JK hoạt động tại sườn âm của xung nhịp. Nhìn vào đồ hình trạng thái ta thấy ở trạng thái trong S2 (Q_0Q_1 = 11) khi có xung nhịp C thì mạch sẽ đưa ra tín hiệu Z = 1.



Hình 5-17 c). Đồ hình trạng thái

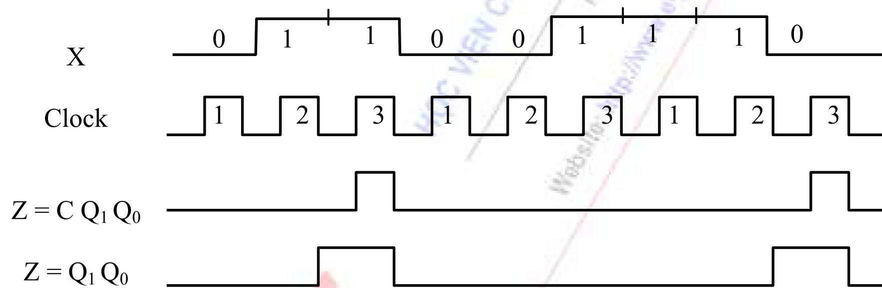
Bước 6: Chức năng của mạch:Trên đồ hình trạng thái ta thấy có hai đường chuyển đổi trạng thái là $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$ và $S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_0$. Theo đường $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$ thì tín hiệu ra Z = 1 sẽ được đưa ra cùng thời điểm có xung nhịp thứ 3. Theo đường $S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_0$ thì không có tín hiệu ra (Z = 0). Do vậy ta sẽ phân tích theo con đường thứ nhất $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$: Sự chuyển đổi trạng thái đầu tiên từ $S_0 \rightarrow S_1$ chỉ nhờ tác động của xung nhịp mà không phụ

thuộc vào trạng thái của X. Chuyển đổi trạng thái thứ hai từ $S_1 \rightarrow S_2$ nhờ tác động của xung nhịp và sự tác động của tín hiệu vào X = 1. Còn sự chuyển đổi trạng thái thứ ba từ $S_2 \rightarrow S_0$ chỉ nhờ tác động của xung nhịp mà không phụ thuộc vào tín hiệu vào.

Như vậy, mạch chỉ đưa ra tín hiệu ra Z = 1 khi đường chuyển đổi đi qua S_2 tức là mạch chỉ đưa ra tín hiệu ra Z = 1 khi dãy tín hiệu vào X có dạng 010, 011, 110 và 111. Có thể biểu diễn dãy tín hiệu vào để mạch có tín hiệu ra Z = 1 như sau:

$$\begin{array}{ccc} & \mathbf{0} & \\ & \downarrow & \\ \mathbf{0} \leftarrow & \mathbf{1} & \rightarrow \mathbf{1} \end{array} \qquad \begin{array}{ccc} & \mathbf{1} & \\ & \downarrow & \\ \mathbf{0} \leftarrow & \mathbf{1} & \rightarrow \mathbf{1} \end{array}$$

Tóm lại, mạch cho ở sơ đồ trên có chức năng kiểm tra dãy tín hiệu vào X ở dạng chuỗi có độ dài bằng 3. Nếu chuỗi tín hiệu vào có dạng là 1 trong 4 dãy: 010, 011, 110 và 111 mạch sẽ cho tín hiệu ra Z = 1 tại thời điểm có xung nhịp thứ 3. Độ rộng của tín hiệu ra Z bằng độ rộng xung nhịp ($Z = C\,Q_1\,Q_0$).



Hình 5-17d) Dạng xung ra của mạch

Hình 5-17 a, b, c, d . Phân tích mạch tuần tự đồng bộ

5.5.2. Thiết kế mạch tuần tự đồng bộ.

5.5.2.1. Các bước thiết kế mạch tuần tự đồng bộ.

Bước 1: Xác định bài toán, gán hàm và biến, tìm hiểu mối quan hệ giữa chúng.

Bước 2: Xây dựng đồ hình trạng thái, bảng chuyển đổi trạng thái và hàm ra.

Bước 3: Rút gọn trạng thái (tối thiểu hoá trạng thái).

Việc tối thiểu hoá trạng thái chủ yếu dựa vào khái niệm trạng thái tương đương. Các trạng thái tương đương với nhau có thể được thay bằng một trạng thái chung đại diện cho chúng.

Bước 4: Mã hoá trạng thái.

Số biến nhị phân dùng để mã hoá các trạng thái trong của mạch phụ thuộc vào số lượng trạng thái trong của mạch. Nếu số lượng trạng thái trong là N, số biến nhị phân cần dùng là *n* thì *n* phải thoả mãn điều kiện: $n \geq \log_2 N$.

Có rất nhiều cách mã hoá khác nhau, mỗi cách cho một sơ đồ thực hiện mạch khác nhau. Vấn đề là phải mã hoá sao cho sơ đồ mạch thực hiện là đơn giản nhất.

Bước 5: Xác định hệ phương trình của mạch. Có hai cách xác định:

+ Lập bảng chuyển đổi trạng thái và tín hiệu ra, từ đó xác định các phương trình kích cho các trigơ.

+ Dựa trực tiếp vào đồ hình trạng thái, viết hệ phương trình T_{on}, T_{off} của các trigơ và phương trình hàm ra.

Bước 6: Vẽ sơ đồ thực hiện.

_5.5.2.2. Ví dụ._

Thiết kế mạch tuần tự thực hiện nhiệm vụ kiểm tra dãy tín hiệu vào ở dạng nhị phân có độ dài bằng 3 được đưa vào liên tiếp trên đầu vào X. Nếu dãy tín hiệu vào có dạng là 010 hoặc 011 hoặc 110 hoặc 111 thì Z = 1. Các trường hợp khác Z = 0.

Bước 1: Xác định bài toán. Mạch được thiết kế có nhiệm vụ phát hiện tín hiệu vào. Khi nhận được 1 trong các dãy tín hiệu trên thì mạch sẽ báo rằng đã nhận được.

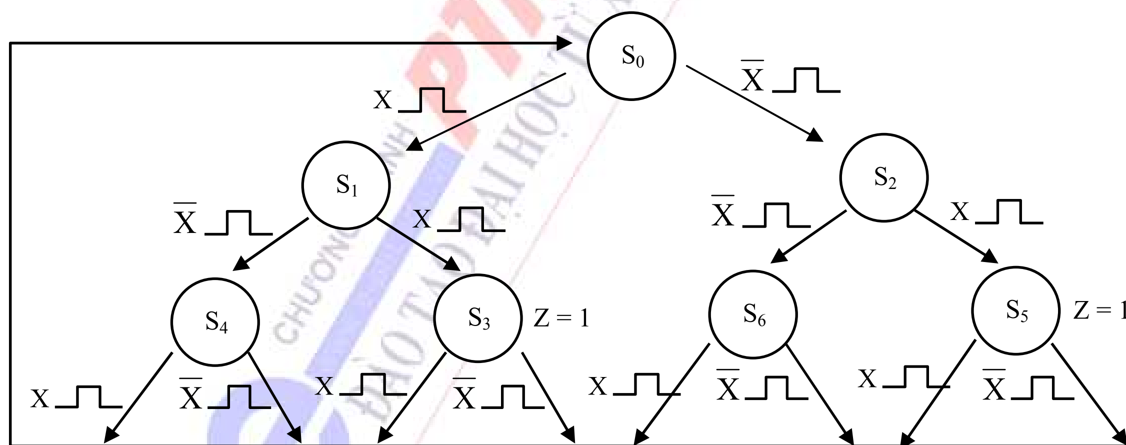
Mạch phải thiết kế là mạch đồng bộ, nên sẽ có các lối vào là X- tín hiệu vào, Ck- xung nhịp điều khiển, Z – tín hiệu ra.

Bước 2: Xây dựng đồ hình trạng thái, bảng chuyển đổi trạng thái

Giả sử trạng thái ban đầu là S_0:

Khi tín hiệu vào là X. Ck thì mạch sẽ chuyển tới trạng thái S_1. Khi tín hiệu vào là $\overline{X}$. Ck mạch sẽ chuyển đến trạng thái S_2.

Tương tự như vậy. Khi mạch ở trạng thái S_1 thì khi có tín hiệu X. Ck mạch chuyển đến trạng thái S_3 và chuyển đến trạng thái S_4 khi có tín hiệu $\overline{X}$. Ck. Tương tự ta xây dựng được đồ hình sau 5-18 a.



Hình 5-18 a). Đồ hình trạng thái

Nếu mạch ở 1 trong 4 trạng thái S_3, S_4, S_5, S_6: khi có tín hiệu vào X. Ck hoặc $\overline{X}$. Ck thì mạch sẽ chuyển về trạng thái ban đầu S_0. Khi dãy tín hiệu vào là 110 hoặc 111 (ứng với đường chuyển đổi trạng thái là $S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_0$) hay khi dãy tín hiệu vào là 010 hoặc 011 (ứng với đường chuyển đổi trạng thái là $S_0 \rightarrow S_3 \rightarrow S_5 \rightarrow S_0$) thì mạch sẽ cho tín hiệu ra Z = 1 tại thời điểm xung thứ 3. Với các đường chuyển đổi khác Z = 0.

Từ đồ hình trạng thái ta xây dựng được bảng chuyển đổi trạng thái như sau:

S	S^k		Z	
	X = 0	X = 1	X = 0	X = 1
S_0	S_2	S_1	0	0
S_1	S_4	S_3	0	0
S_2	S_6	S_5	0	0
S_3	S_0	S_0	1	1
S_4	S_0	S_0	0	0
S_5	S_0	S_0	1	1
S_6	S_0	S_0	0	0

Hình 5-18b). Bảng chuyển đổi trạng thái

Bước 3: Tối thiểu hoá trạng thái. Để có được sơ đồ mạch đơn giản ta phải tối thiểu hoá các trạng thái. Trong phần này sẽ giới thiệu phương pháp tối thiểu hoá Caldwell. Cơ sở lý thuyết của việc tối thiểu hoá là dựa vào khái niệm các trạng thái tương đương.

Định nghĩa các trạng thái tương đương:

Trạng thái S_i được gọi là trạng thái tương đương với trạng thái S_j ($S_i \approx S_j$) khi và chỉ khi: nếu lấy S_i và S_j là hai trạng thái ban đầu thì với mọi dãy tín hiệu vào có thể chúng luôn cho dãy tín hiệu ra giống nhau.

Nếu có nhiều trạng thái tương đương với nhau từng đôi một thì chúng tương đương với nhau (tính chất bắc cầu). Để kiểm tra một nhóm các trạng thái xem chúng có tương đương với nhau không, có thể sử dụng bảng trạng thái và tín hiệu ra như sau:

- Nhóm các trạng thái tương đương phải có những hàng trong bảng tín hiệu ra giống nhau.
- Nhóm các trạng thái tương đương phải có những hàng trong bảng trạng thái ở cùng một cột (ứng với cùng một tổ hợp tín hiệu vào) là tương đương. Nghĩa là ứng với cùng một tổ hợp tín hiệu vào các trạng thái kế tiếp của chúng là tương đương.

Quy tắc Caldwell:

Những hàng (tương ứng với trạng thái trong) của bảng chuyển đổi trạng thái và tín hiệu ra sẽ được kết hợp với nhau và được biểu diễn bằng một hàng chung - đặc trưng (trạng thái đặc trưng) cho chúng nếu như chúng thoả mãn hai điều kiện sau:

1. Các hàng tương ứng trong ma trận ra giống nhau.
2. Trong ma trận ra, các hàng tương ứng phải thoả mãn 1 trong 3 điều sau:

- Các hàng trong ma trận trạng thái giống nhau.
- Các trạng thái ở trong cùng một cột nằm trong nhóm trạng thái được xét.
- Các trạng thái ở trong cùng một cột là các trạng thái tương đương.

Sau khi đã thay thế các trạng thái tương đương bằng một trạng thái chung đặc trưng cho chúng, lặp lại các công việc tìm các trạng thái tương đương khác cho đến khi không thể tìm được

các trạng thái tương đương nào nữa thì dừng lại. Số trạng thái trong bảng chuyển đổi trạng thái là tối thiểu.

Nhược điểm của phương pháp này là khi số trạng thái quá lớn thì công việc tối thiểu hoá mất nhiều thời gian.

Áp dụng quy tắc Caldwell cho bài toán trên ta thấy trạng thái S_4 tương đương với trạng thái S_6 ($S_4 \approx S_6$), S_3 tương đương với S_5 ($S_3 \approx S_5$). Thay thế các trạng thái tương đương bằng một trạng thái chung đặc trưng cho chúng. Ví dụ thay thế S_4, S_6 bằng S_{46}, thay thế S_3, S_5 bằng S_{35}. Từ đó lập được bảng chuyển đổi trạng thái 5-18c) và 5-18 d):

S \ X	0	1
S_0	S_2 Z = 0	S_1 Z = 0
S_1	S_{46} Z = 0	S_{35} Z = 0
S_2	S_{46} Z = 0	S_{35} Z = 0
S_{35}	S_0 Z = 1	S_0 Z = 1
S_{46}	S_0 Z = 0	S_0 Z = 0

Hình 5-18c) Bảng chuyển đổi trạng thái sau khi gộp S_3 và S_5, S_4 và S_6

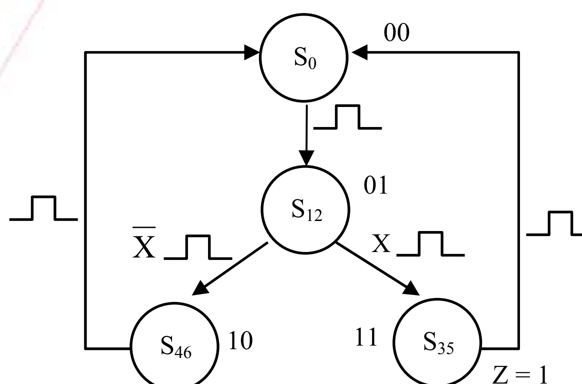
S \ X	0	1
S_0	S_{12} Z = 0	S_{12} Z = 0
S_{12}	S_{46} Z = 0	S_{35} Z = 0
S_{35}	S_0 Z = 1	S_0 Z = 1
S_{46}	S_0 Z = 0	S_0 Z = 0

Hình 5-18d) Bảng chuyển đổi trạng thái sau khi gộp S_1 và S_2

Bước 4: Sau khi gộp hai trạng thái S_1 và S_2 thành trạng thái chung S_{12} thì mạch chỉ còn 4 trạng thái S_0, S_{12}, S_{35}, S_{46}. Mã hoá 4 trạng thái này bằng hai biến nhị phân Q_1 và Q_0.

Q_0	Q_1	Mã hoá S
0	0	S_0
0	1	S_{12}
1	1	S_{35}
1	0	S_{46}

Hình 5-18 e) Bảng mã hoá trạng thái



Hình 5-18f). Đồ hình trạng thái tối giản

Bước 5: Xác định hệ phương trình của mạch.

Có hai cách xác định hệ phương trình này.

Cách 1:

Dựa vào bảng chuyển đổi trạng thái ta lập bảng hàm kích 5-13 cho hai trigơ Q_0 và Q_1.

Dùng bảng Karnaugh 5-14 để rút gọn, ta thu được kết quả sau:

$J_0 = Q_1$; $K_0 = 1$

$J_1 = \overline{Q_0}$; $K_1 = \overline{X} + Q_0$

$Z = X\, Q_0 Q_1$

Trạng thái hiện tại	**Trạng thái kế tiếp**		**Các đầu vào của trigơ**							
	X = 0	**X = 1**	**X = 0**		**X = 1**		**X = 0**		**X = 1**	
Q_0Q_1	**Q_0Q_1**	**Q_0Q_1**	**J_0**	**K_0**	**J_0**	**K_0**	**J_1**	**K_1**	**J_1**	**K_1**
00	**01** **Z = 0**	**01** **Z = 0**	**0**	**X**	**0**	**X**	**1**	**X**	**1**	**X**
01	**10** **Z = 0**	**11** **Z = 0**	**1**	**X**	**1**	**X**	**X**	**1**	**X**	**0**
11	**00** **Z = 1**	**00** **Z = 1**	**X**	**1**	**X**	**1**	**X**	**1**	**X**	**1**
10	**00** **Z = 0**	**00** **Z = 0**	**X**	**1**	**X**	**1**	**0**	**X**	**0**	**X**

Bảng 5-13. Bảng hàm kích thích

X \ Q_0Q_1	00	01	11	10
0		1	x	x
1			x	x

$J_0 = Q_1$

X \ Q_0Q_1	00	01	11	10
0	x	x	1	x
1	x	x	1	x

$K_0 = 1$

X \ Q_0Q_1	00	01	11	10
0	1	x	x	
1	1	x	x	

$J_1 = \overline{Q_0}$

X \ Q_0Q_1	00	01	11	10
0	x	1	1	x
1	x		1	x

$K_1 = \overline{X} + Q_0$

X \ Q_0Q_1	00	01	11	10
0			1	
1			1	

$Z = X\, Q_0 Q_1$

Bảng 5-14. Bảng tính hàm kích

Cách 2: Dựa trực tiếp vào đồ hình trạng thái viết phương trình T_{on}, T_{off} của từng trigơ và phương trình tín hiệu ra.

Đối với trigơ JK nếu:

$T_{on\,Q} = T^{*}\,\overline{Q} \quad \Rightarrow J_Q = T^{*}$

$T_{offQ} = T^{**}\,Q \quad \Rightarrow K_Q = T^{**}$

Đối với trường hợp này ta có:

$$T_{onQ_0} = S_{12}X + S_{12}\overline{X} = S_{12} = \overline{Q_0}Q_1 \qquad \Rightarrow J_0 = Q_1$$

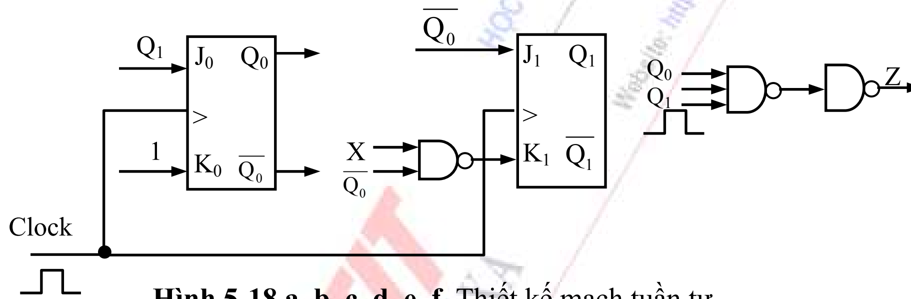
$$T_{offQ_0} = S_{35} + S_{46} = Q_0Q_1 + Q_0\overline{Q_1} = Q_0 \qquad \Rightarrow K_0 = 1$$

$$T_{onQ_1} = S_0X = \overline{Q_0}\,\overline{Q_1} \qquad \Rightarrow J_1 = \overline{Q_0}$$

$$T_{offQ_1} = S_{12}\overline{X} + S_{35} = \overline{Q_0}\,Q_1\,\overline{X} + Q_0Q_1 = Q_1(\overline{Q_0}\,\overline{X} + Q_0) \qquad \Rightarrow K_1 = \overline{Q_0}\,\overline{X} + Q_0 = \overline{X} + Q_0$$

Phương trình hàm ra $Z = Q_0Q_1Ck$

Bước 6: Sơ đồ mạch điện:



Hình 5-18 a, b, c, d, e, f. Thiết kế mạch tuần tự

5.6. MẠCH TUẦN TỰ KHÔNG ĐỒNG BỘ

Phần 5.6 đã nghiên cứu các mạch tuần tự đồng bộ, hoạt động của chúng được điều khiển bởi các xung nhịp. Nhưng trên thực tế có nhiều mạch lại được điều khiển bởi các sự kiện mà không tuân theo một quy luật nào cả. Ví dụ một hệ thống chống trộm sẽ chỉ hoạt động khi có trộm. Những mạch tuần tự hoạt động theo kiểu như vậy gọi là mạch tuần tự không đồng bộ.

Mạch tuần tự không đồng bộ có thể thiết kế:

- Chỉ dùng những mạch NAND.

- Dùng trigơ RS không đồng bộ và các mạch NAND.

Việc thiết kế mạch tuần tự không đồng bộ dùng các trigơ loại không đồng bộ khác hoàn toàn tương tự.

5.6.1. Các bước thiết kế mạch tuần tự không đồng bộ

Bước 1: Xác định bài toán, gán hàm và biến, tìm hiểu mối quan hệ giữa chúng.

Bước 2: Xây dựng đồ hình trạng thái, bảng chuyển đổi trạng thái và hàm ra.

Bước 3: Rút gọn trạng thái (tối thiểu hoá trạng thái).

Việc tối thiểu hoá trạng thái chủ yếu dựa vào khái niệm trạng thái tương đương. Các trạng thái tương đương với nhau có thể được thay bằng một trạng thái chung đại diện cho chúng.

Bước 4: Mã hoá trạng thái.

Số biến nhị phân dùng để mã hoá các trạng thái trong của mạch phụ thuộc vào số lượng trạng thái trong của mạch. Nếu số lượng trạng thái trong là N, số biến nhị phân cần dùng là n thì n phải thoả mãn điều kiện: $n \geq \log_2 N$.

Có rất nhiều cách mã hoá khác nhau, mỗi cách cho một sơ đồ thực hiện mạch khác nhau. Vấn đề là phải mã hoá sao cho sơ đồ mạch thực hiện là đơn giản nhất.

Do mạch không đồng bộ hoạt động không có sự tác động của xung nhịp cho nên trong mạch thường có các hiện tượng chạy đua làm cho hoạt động của mạch bị sai, vì vậy khi mã hoá trạng thái phải tránh hiện tượng này.

Bước 5: Xác định hệ phương trình của mạch. Có hai cách xác định:

+ Lập bảng chuyển đổi trạng thái và tín hiệu ra, từ đó xác định các phương trình kích cho các trigơ.

+ Dựa trực tiếp vào đồ hình trạng thái, viết hệ phương trình T_{on}, T_{off} của các trigơ và phương trình hàm ra.

Cả hai cách này đều có hai loại phương trình:

- Phương trình của mạch chỉ dùng NAND.
- Phương trình của mạch dùng trigơ RS không đồng bộ và các mạch NAND

Bước 6: Vẽ sơ đồ thực hiện.

Sau đây là nội dung của từng phương pháp.

Cách 1: Dựa vào bảng chuyển đổi trạng thái.

a) Chỉ dùng các mạch NAND

Ký hiệu : *A, B, …N* là các biến nhị phân dùng để mã hoá các trạng thái trong của mạch.

$X_1, X_2 \ldots X_m$ là các tín hiệu vào đã được mã hoá nhị phân.

$Z_1, Z_2 \ldots Z_m$ là các tín hiệu ra đã được mã hoá nhị phân.

Dựa vào bảng chuyển đổi trạng thái xác định hệ phương trình:

$A^k = f_A (A, B, \ldots N, X_1, X_2 \ldots X_m)$

$B^k = f_B (A, B, \ldots N, X_1, X_2 \ldots X_m)$

……

$N^k = f_N (A, B, \ldots N, X_1, X_2 \ldots X_m)$

$Z_1 = g_1 (A, B, \ldots N, X_1, X_2 \ldots X_m)$

$Z_2 = g_2 (A, B, \ldots N, X_1, X_2 \ldots X_m)$

……

$Z_n = g_n(A, B, \ldots N, X_1, X_2 \ldots X_m)$

Tối thiểu hoá hệ hàm và viết phương trình ở dạng chỉ dùng NAND.

b) Mạch dùng trigơ RS và các mạch NAND

Trong bảng trạng thái căn cứ vào sự thay đổi trạng thái của từng trigơ:

$A \Rightarrow A^k$, $B \Rightarrow B^k, \ldots, N \Rightarrow N^k$, xác định được giá trị tương ứng của đầu vào kích R, S cho từng trigơ, từ đó viết được hệ phương trình:

$R_A = \Phi_1(A, \ldots N, X_1, X_2 \ldots X_m)$

$S_A = \Phi_2(A, \ldots N, X_1, X_2 \ldots X_m)$

Tối thiểu hoá các hàm và viết phương trình ở dạng chỉ dùng NAND.

Tương tự với $B, C, \ldots N$ cũng như vậy.

Ta xác định tín hiệu ra :

$Z = \Phi(A, \ldots N, X_1, X_2 \ldots X_m)$

Tối thiểu hoá và viết phương trình ở dạng chỉ dùng NAND.

Cách 2: Dựa trực tiếp vào đồ hình trạng thái

Ta có phương trình đầu vào kích (R, S) của trigơ A là:

S_A = tập hợp bật của A + [(1)]

R_A = tập hợp tắt của A + [(0)]

Làm tương tự với các trigơ khác.

a) Chỉ dùng mạch NAND

Ta có phương trình đặc trưng của trigơ RS

$$Q^k = S + \overline{R}Q \qquad \Rightarrow A^k = S_A + \overline{R_A}\,A$$

Sau đó ta phải tối thiểu hoá phương trình và viết dưới dạng chỉ dùng NAND. Đối với các trigơ khác cũng làm như vậy.

b) Dùng các trigơ RS không đồng bộ và các mạch NAND

$R_A = \Phi_{1A}(A, \ldots N, X_1, X_2 \ldots X_m)$

$S_A = \Phi_{2A}(A, \ldots N, X_1, X_2 \ldots X_m)$

……

$R_N = \Phi_{1N}(A, \ldots N, X_1, X_2 \ldots X_m)$

$S_N = \Phi_{2N}(A, \ldots N, X_1, X_2 \ldots X_m)$

$Z_1 = \Psi_1(A, \ldots N, X_1, X_2 \ldots X_m)$

$Z_2 = \Psi_2(A, \ldots N, X_1, X_2 \ldots X_m)$

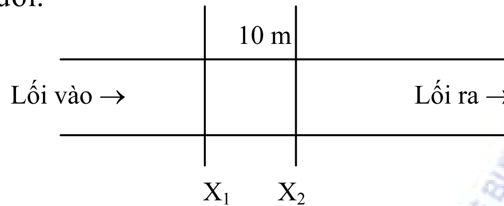
……

$Z_n = \Psi_n (A\ ,\ ...N\ , X_1, X_2...X_m\ )$

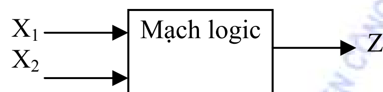
Tối thiểu hoá hệ phương trình.

5.6.2. Ví dụ

Một mạch tuần tự không đồng bộ được thiết kế để đếm số người vào thăm một viện bảo tàng. Mạch gồm hai đèn X_1, X_2 được bố trí cách nhau 10 mét. Mạch được thiết kế sao cho mỗi lần chỉ đếm được một người.



Hình 5-19 a) Bố trí các đèn ở của vào viện bảo tàng



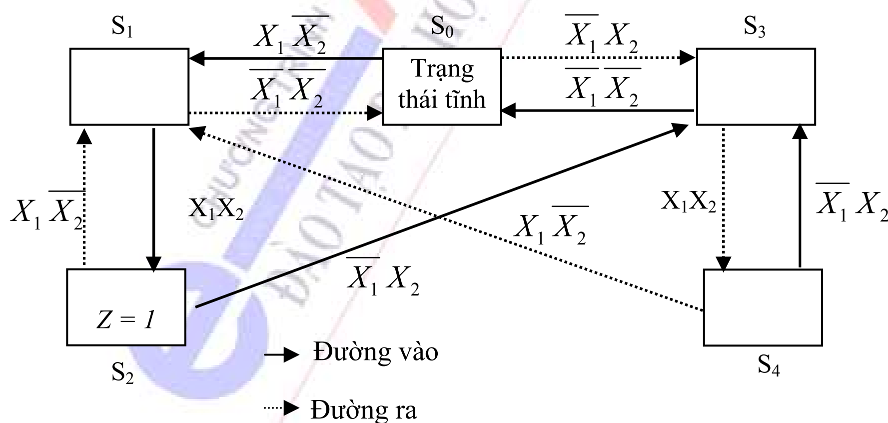
Hình 5-19 b) Sơ đồ khối của mạch

Khi có một người đi vào thì hai đèn sẽ bị chắn liên tiếp. Đầu tiên X_1 bị chắn, tiếp đến cả X_1 và X_2 cùng bị chắn, sau đó đến X_2 bị chắn. Khi đó mạch cho ra tín hiệu Z = 1. Khi một người ra thì sẽ ngược lại. Đầu tiên đèn X_2 sẽ bị chắn, sau đó cả X_1 và X_2 cùng bị chắn và cuối cùng chỉ có X_1 bị chắn. Sơ đồ khối của mạch tạo tín hiệu đếm Z được mô tả bởi hình 5-19b.

Hai lối vào của mạch là X_1 X_2. Đầu ra Z được đưa tới lối vào của bộ giải mã.

Ta quy ước: đèn bị chắn = X; ngược lại thì = $\overline{X}$.

Đồ hình trạng thái được mô tả ở hình 5-19c.



Hình 5-19 c)Đồ hình trạng thái

S_0 là trạng thái ban đầu của mạch. Nếu một người đi vào thì sự chuyển đổi của mạch sẽ là $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_0$. Nếu một người đi ra thì quá trình chuyển đổi trạng thái của mạch là $S_0 \rightarrow S_3 \rightarrow S_4 \rightarrow S_1 \rightarrow S_0$. Khi có một người ngập ngừng sau đó lại quay ra ban đầu chắn đèn X_1 sau đó quay ra thì mạch sẽ chuyển đổi trạng thái $S_0 \rightarrow S_1 \rightarrow S_0$, lúc đó mạch sẽ không thực hiện đếm.

Tương ứng với đồ hình trạng thái trên ta lập được bảng chuyển đổi trạng thái ở hình 5-19d:

Trạng thái hiện tại	**Trạng thái kế tiếp và tín hiệu ra**			
	X_1 X_2 0 0	**X_1 X_2 0 1**	**X_1 X_2 1 1**	**X_1 X_2 1 0**
S_0	**S_0 / Z = 0**	**S_3 / Z = 0**		**S_1 / Z = 0**
S_1	**S_0 Z = 0**		**S_2 Z = 0**	**S_1 / Z = 0**
S_2		**S_3 / Z = 0**	**S_2 / Z = 1**	**S_1 / Z = 0**
S_3	**S_0 / Z = 0**	**S_3 / Z = 0**	**S_4 / Z = 0**	
S_4		**S_3 / Z = 0**	**S_4 / Z = 0**	**S_1 / Z = 0**

Hình 5-19 d) Bảng chuyển đổi trạng thái và hàm ra

Bảng có 5 hàng ứng với 5 trạng thái hiện tại có thể xuất hiện và 4 cột, mỗi cột ứng với một tổ hợp giá trị có thể của X_1, X_2. Mỗi ô của bảng biểu diễn trạng thái kế tiếp và tín hiệu ra tương ứng với trạng thái hiện tại và giá trị của tín hiệu vào X_1, X_2.

Trong bảng chuyển đổi trạng thái, những ô được khoanh tròn là những ô có trạng thái kế tiếp bằng trạng thái hiện tại. Những trạng thái đó là những trạng thái ổn định. Điều kiện cho trạng thái ổn định là $S^k = S$.

Trên bảng có những ô trống. Những ô này tương ứng với các tổ hợp tín hiệu không xuất hiện ở đầu vào. Những ô này có thể điền giá trị tuỳ chọn để tối thiểu hoá hệ phương trình của mạch.

Tiến hành tối thiểu hoá:

Có thể gán trạng thái kế tiếp và tín hiệu ra vào các ô trống sao cho hàng có ô trống có thể kết hợp với các hàng khác.

Ở bảng chuyển đổi trạng thái các hàng S_0, S_1, S_2, và S_3, S_4 có các trạng thái kế tiếp và tín hiệu ra tương ứng là giống nhau nếu như ta gán:

- ô trống của hàng đầu tiên (ứng với S_0) là S_2 / Z = 1,

- ô trống của hàng thứ hai là S_3 / Z = 0,

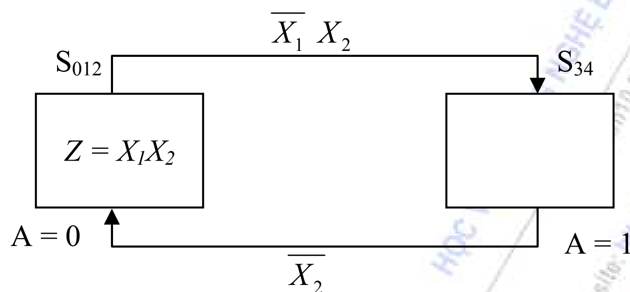
- ô trống của hàng thứ tư là S_1 / Z = 0,

- ô trống của hàng thứ ba và thứ năm là S_0 / Z = 0,

Khi đó bảng chuyển đổi trạng thái được rút gọn lại như sau:

Trạng thái hiện tại	Trạng thái kế tiếp và tín hiệu ra			
	X_1 X_2 0 0	X_1 X_2 0 1	X_1 X_2 1 1	X_1 X_2 1 0
S_{012}	(S_{012}) / Z = 0	S_{34} / Z = 0	(S_{012}) / Z = 0	(S_{012}) / Z = 0
S_{34}	S_{012} / Z = 0	(S_{34}) / Z = 0	(S_{34}) / Z = 0	S_{012} / Z = 0

Hình 5-19 e) Bảng chuyển đổi trạng thái và hàm ra rút gọn



Hình 5-19 f) Đồ hình trạng thái sau khi rút gọn

Mạch chỉ có hai trạng thái nên để mã hoá ta chỉ cần sử dụng một biến nhị phân A. Để mã hoá trạng thái S_{012} thì A = 0, S_{34} thì A = 1. Tín hiệu ra Z = 1 ở trạng thái S_{012} khi X_1 X_2 = 11.

Ta dùng trigơ RS để thiết kế (dựa vào bảng hàm kích của trigơ RS-bảng 5-15).

Ta có phương trình đầu vào kích (*R, S*) của trigơ là:

S = tập hợp bật của *Q* + [(1)] ; Tập hợp bật của Q (T_{on}) là các cung mà Q chuyển từ 0 → 1.

Cung	Q	Q^k	R	S
0	0	0	X	0
T_{on}	0	1	0	1
T_{off}	1	0	1	0
1	1	1	0	X

Bảng 5-15.Bảng hàm kích

$$S_A = \overline{A}\,\overline{X_1}\,X_2 + A\,\overline{X_1}\,X_2 = \overline{X_1}\,X_2$$

R = tập hợp tắt của *Q* + [(0)] ; Tập hợp tắt của Q (T_{off}) là các cung mà Q chuyển từ 1 → 0.

$$R_A = A\overline{X_2} + \overline{A}\,\overline{X_1}\,\overline{X_2} + \overline{A}\,X_1\,\overline{X_2} = \overline{X_2}$$

Các cung [(0)], [(1)] được lấy giá trị không xác định (x) và được dùng để tối thiểu hoá.

Phương trình đặc trưng của trigơ RS

$$Q^k = S_A + \overline{R_A}\,Q_A$$

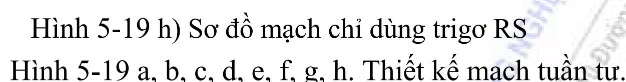
Thay giá trị của R_A, S_A vào biểu thức thu được kết quả:

$$A^k = \overline{X_1}\,X_2 + X_2A = \overline{\overline{\overline{X_1}\,X_2 + X_2A}} = \overline{\overline{\overline{X_1}\,X_2}\,.\,\overline{X_2A}}$$

Phương trình ra:



Hình 5-19 g) Sơ đồ mạch chỉ dùng NAND



Hình 5-19 h) Sơ đồ mạch chỉ dùng trigơ RS

Hình 5-19 a, b, c, d, e, f, g, h. Thiết kế mạch tuần tự.

Nếu thiết kế mạch dùng trigơ RS và các mạch NAND ta có:

$$S_A = \overline{X_1} X_2 \qquad R_A = \overline{X_2}$$

Và mạch được biểu diễn ở hình 5-19 g, h.

5.7. HIỆN TƯỢNG CHU KỲ VÀ CHẠY ĐUA TRONG MẠCH KHÔNG ĐỒNG BỘ

Đối với mạch tuần tự đồng bộ, việc mã hoá trạng thái là làm sao cho sơ đồ thực hiện mạch là đơn giản nhất.

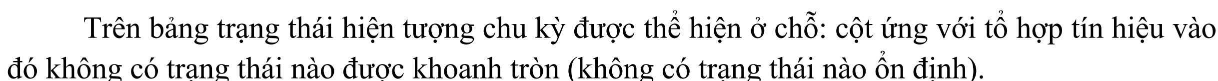
Đối với mạch tuần tự không đồng bộ, trong mạch thường xảy ra các hiện tượng hoặc là chu kỳ hoặc là chạy đua. Những hiện tượng này làm cho mạch hoạt động sai lệch đi so với chức năng của nó. Vì vậy, khi mã hoá trạng thái của mạch tuần tự không đồng bộ ta phải tránh các trường hợp đó.

5.7.1. Hiện tượng chu kỳ trong mạch tuần tự không đồng bộ.

Định nghĩa:

Hiện tượng chu kỳ là hiện tượng tại một tổ hợp tín hiệu vào nào đó, mạch liên tục chuyển từ trạng thái này sang trạng thái khác theo một chu kỳ kín. Nghĩa là trong quá trình đó không có trạng thái nào ổn định. Do vậy, khi thay đổi tín hiệu vào không xác định được mạch đang ở trạng thái nào trong dãy trạng thái nói trên.

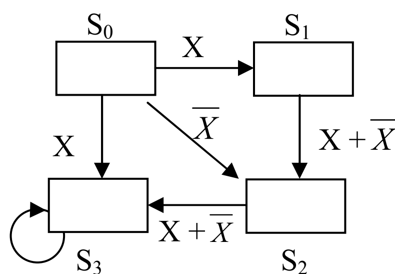
Ví dụ: ứng với một tổ hợp tín hiệu vào quá trình chuyển đổi trạng thái theo chu trình sau:



Trên bảng trạng thái hiện tượng chu kỳ được thể hiện ở chỗ: cột ứng với tổ hợp tín hiệu vào đó không có trạng thái nào được khoanh tròn (không có trạng thái nào ổn định).

Ví dụ: Đồ hình trạng thái của một mạch tuần tự không đồng bộ được biểu diễn trên hình 5-20a. Việc mã hoá trạng thái sử dụng biến nhị phân A và B là tuỳ chọn. Từ đồ hình trạng thái ta lập bảng chuyển đổi trạng thái 5-20b.

Giả thiết ban đầu mạch ở trạng thái S_3 (AB = 10) và X = 0. Sau đó tín hiệu vào X thay đổi từ 0 đến 1 thì mạch sẽ chuyển trạng thái từ S_3 sang S_0. Nếu X vẫn bằng 1 thì mạch sẽ lần lượt chuyển đến các trạng thái tiếp theo là S_1, S_2, …S_0. Khi X = 1 chu trình chuyển đổi trạng thái như hình 5-21:

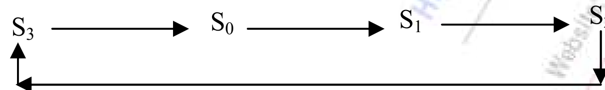


Hình 5-20a) Đồ hình trạng thái

S \ X	0	1
S_0	S_2	S_1
S_1	S_2	S_2
S_2	S_3	S_3
S_3	S_3	S_0

Hình 5-20b) Bảng chuyển đổi trạng thái

Hình 5-20 a,b. Bảng trạng thái hiện tượng chu kỳ



Hình 5-21. Chu trình chuyển đổi trạng thái

Khi đó mạch không có trạng thái ổn định.

5.7.2. Hiện tượng chạy đua trong mạch tuần tự không đồng bộ.

Định nghĩa:

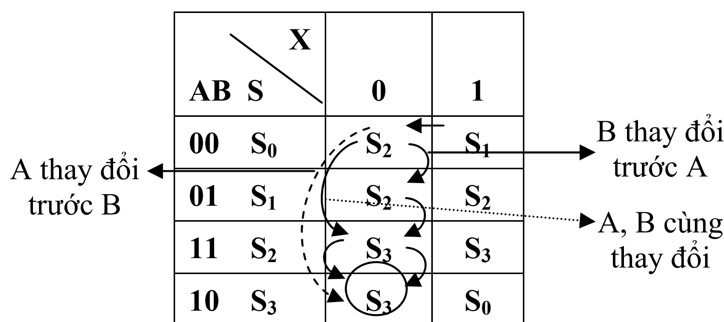
Hiện tượng chạy đua trong mạch không đồng bộ là hiện tượng: do tính không đồng nhất của các phần tử nhị phân dùng để mã hoá trạng thái, vì mạch hoạt động không đồng bộ, khi mạch chuyển trạng thái từ $S_i \rightarrow S_j$ mạch có thể chuyển biến trạng thái theo những con đường khác nhau.

Nếu trạng thái cuối cùng của những con đường đó là ổn định và duy nhất thì chạy đua không nguy hiểm. Ngược lại, chạy đua nguy hiểm là những cách chuyển biến trạng thái khác nhau đó cuối cùng dẫn đến các trạng thái ổn định khác nhau, có thể tới trạng thái khoá và không thoát ra được.

Ví dụ: Chạy đua không nguy hiểm: Một mạch tuần tự không đồng bộ có bảng trạng thái mô tả ở hình 5-22 .

Nhìn vào bảng ta thấy nếu mạch đang ở trạng thái S_0 (AB = 00) tín hiệu vào X thay đổi từ 0 $\rightarrow$ 1 mạch sẽ chuyển trược tiếp tới trạng thái S_2 (AB = 01) và nếu X vẫn bằng 0 trạng thái tiếp theo của mạch sẽ là S_3, nó sẽ là trạng thái ổn định cuối cùng của mạch nếu như X vẫn bằng 0.

Mạch có thể thay đổi trạng thái theo những con đường khác nhau tuỳ thuộc vào thứ tự thay đổi (hay thời gian quá độ) của A và B



Hình 5-22. Hiện tượng chạy đua không nguy hiểm trong mạch tuần tự không đồng bộ

Nếu A và B thay đổi đồng thời mạch sẽ chuyển trạng thái sang S_2 rồi mới sang S_3.

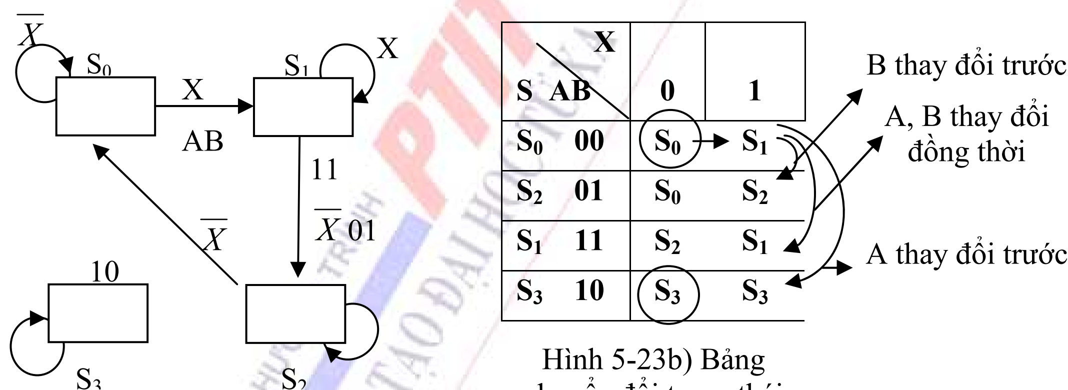
Nếu B thay đổi trước A thì mạch sẽ lần lượt chuyển qua S_1, S_2 rồi mới sang S_3.

Nếu A thay đổi trước B mạch sẽ chuyển đổi từ $S_0 \rightarrow S_3$.

Ta thấy rằng cả ba con đường đều dẫn đến cùng một trạng thái ổn định S_3. Vậy hiện tượng chạy đua này không nguy hiểm.

Khi mạch đang ở trạng thái ổn định (trạng thái được khoanh tròn), nó chỉ thay đổi trạng thái khi tín hiệu vào thay đổi.

Chạy đua nguy hiểm: Đồ hình trạng thái của mạch không đồng bộ mô tả ở hình 5- 23a.



Hình 5-23a) Đồ hình trạng thái

Hình 5-23b) Bảng chuyển đổi trạng thái

Hình 5-23. Hiện tượng chạy đua nguy hiểm trong mạch không đồng bộ

Giả thiết trạng thái ban đầu của mạch là S_0 (AB = 00) và tín hiệu vào X = 0. Nếu X thay đổi từ $0 \rightarrow 1$ thì mạch sẽ chuyển đổi trạng thái như sau:

- Nếu A, B thay đổi đồng thời thì mạch sẽ chuyển đến trạng thái S_1.

- Nếu B thay đổi trước A thì mạch sẽ chuyển đến trạng thái S_2.

- Nếu A thay đổi trước B thì mạch sẽ chuyển đến trạng thái S_3.

Ở đây trạng thái S_3 là trạng thái "khoá". Như vậy khi A thay đổi trước B thì mạch sẽ rơi vào trạng thái khoá và không thoát ra được.

Chạy đua này là chạy đua nguy hiểm.

5.7.3. Tối thiểu hoá và mã hoá trạng thái trong mạch tuần tự không đồng bộ.

5.7.3.1. Tối thiểu hoá trạng thái

Tối thiểu hoá trạng thái là giảm bớt số trạng thái (nếu có thể) để mạch thiết kế là đơn giản và do vậy tin cậy hơn.

Đối với các ô trống trong bảng chuyển đổi trạng thái (những ô này ứng với tổ hợp tín hiệu vào không xuất hiện) có thể lấy giá trị tuỳ chọn để kết quả tối thiểu hoá là tối giản.

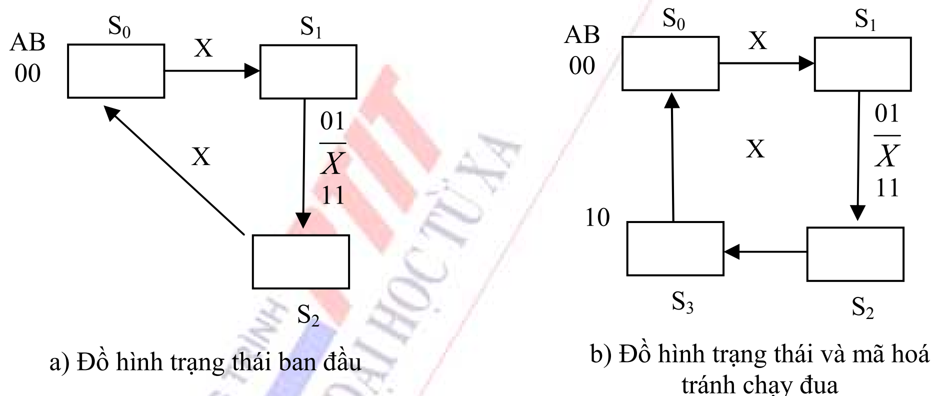
5.7.3.2. Mã hoá trạng thái

Sử dụng các biến nhị phân để mã hoá các trạng thái trong của mạch. Đối với mạch tuần tự không đồng bộ phải mã hoá trạng thái để tránh được hiện tượng chu kỳ và chạy đua.

Để tránh được hiện tượng chu kỳ thì khi có mọi tín hiệu vào nhưng mạch phải luôn có một trạng thái ổn định.

Để tránh hiện tượng chạy đua, phải mã hoá trạng thái sao cho với tất cả các chuyển đổi trạng thái có thể có của mạch chỉ có duy nhất một biến thay đổi.

Ví dụ. Đồ hình trạng thái của mạch tuần tự không đồng bộ được mô tả như hình 5-24:



Hình 5-24. Tránh chạy đua trong mạch không đồng bộ

Cần hai biến nhị phân A và B để mã hoá 3 trạng thái này. Giả sử chọn cách mã hoá như hình 5-24a.

Với cách mã hoá này khi thay đổi từ $S_2 \rightarrow S_0$ cả hai biến A và B đều thay đổi. Điều này dẫn đến hiện tượng chạy đua trong mạch.

Do vậy, để tránh hiện tượng chạy đua đưa thêm một trạng thái giả S_3 để cho thay đổi từ $S_2 \rightarrow S_0$ thông qua trạng thái giả này bảo đảm quá trình thay đổi trạng thái luôn chỉ có một biến thay đổi. Đồ hình này tránh được hiện tượng chạy đua.

Khi sử dụng các trạng thái giả để mã hoá cho mạch cần lưu ý tìm cách cho mạch thoát khỏi các trạng thái giả đó. Phần lớn các trường hợp ta cho mạch thoát khỏi các trạng thái giả đó vô điều kiện.

5.8. MỘT SỐ MẠCH TUẦN TỰ THÔNG DỤNG

5.8.1. Bộ đếm.

Bộ đếm là mạch tuần tự đơn giản, nó được xây dựng từ các phần tử nhớ là các trigơ và các mạch logic tổ hợp.

Các bộ đếm là thành phần cơ bản của các hệ thống số, chúng được sử dụng để đếm thời gian, chia tần số, điều khiển các mạch khác…Bộ đếm được sử dụng rất nhiều trong máy tính, trong thông tin. Để xây dựng bộ đếm, người ta có thể dùng mã nhị phân hoặc các loại mã khác như mã Gray, mã NBCD, mã vòng…

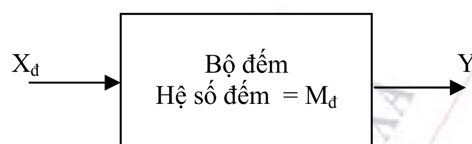
Phần này sẽ đưa ra những đặc điểm cơ bản nhất của bộ đếm và các phương pháp thiết kế bộ đếm.

5.8.1.1. Định nghĩa và phân loại bộ đếm

1. Định nghĩa.

Bộ đếm là một mạch tuần tự tuần hoàn có một lối vào đếm và một lối ra, mạch có số trạng thái trong bằng chính hệ số đếm (ký hiệu là $M_đ$). Dưới tác dụng của tín hiệu vào đếm, mạch sẽ chuyển từ trạng thái trong này đến một trạng thái trong khác theo một thứ tự nhất định. Cứ sau $M_đ$ tín hiệu vào đếm mạch lại trở về trạng thái xuất phát ban đầu.

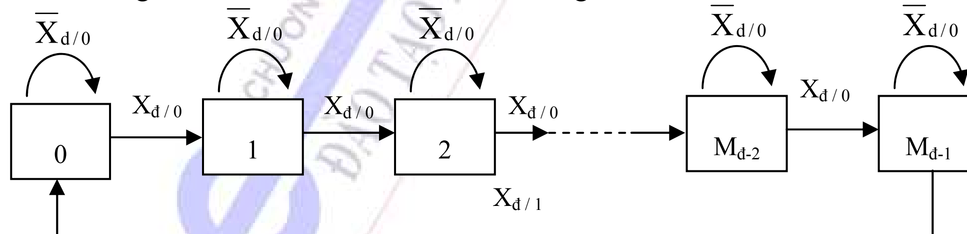
Sơ đồ khối được mô tả như ở hình 5- 25.



Hình 5- 25 Sơ đồ khối của bộ đếm

2. Đồ hình trạng thái tổng quát của bộ đếm.

Đồ hình trạng thái của bộ đếm có hệ số đếm bằng $M_đ$ được mô tả ở hình 5-26.



Hình 5-26. Đồ hình trạng thái của bộ đếm $M_đ$

Khi không có tín hiệu vào đếm ($X_đ$) mạch giữ nguyên trạng thái cũ, khi có tín hiệu đếm thì mạch sẽ chuyển đến trạng thái kế tiếp.

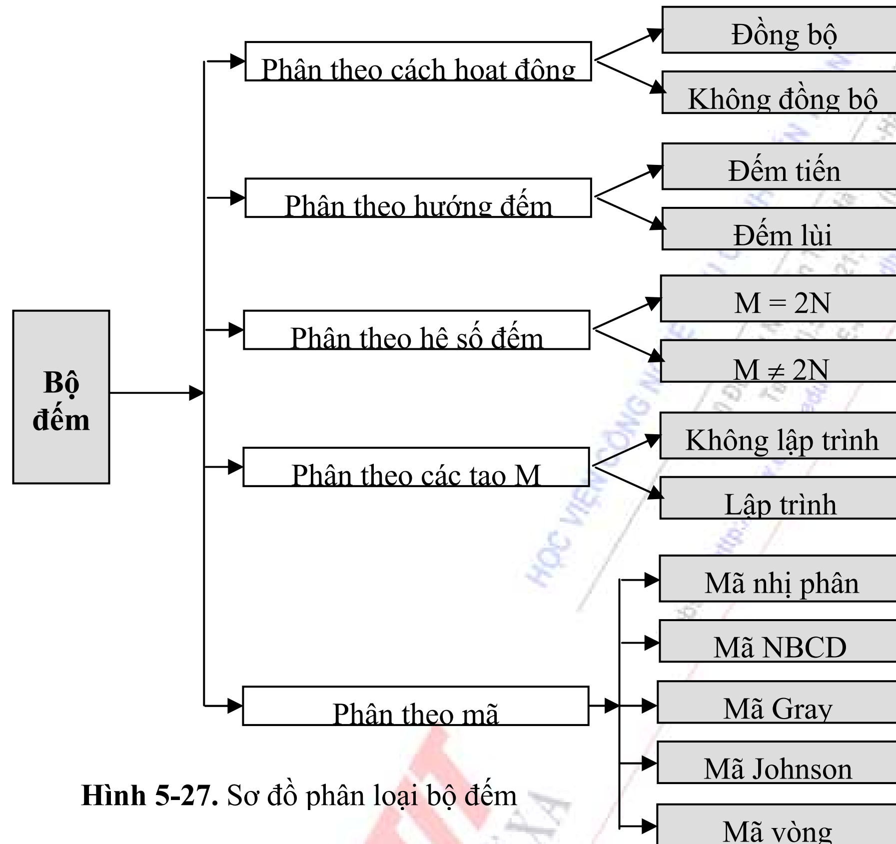
Tính chất tuần hoàn của bộ đếm thể hiện ở chỗ: sau $M_đ$ tín hiệu vào $X_đ$ thì mạch lại quay trở về trạng thái xuất phát ban đầu.

Tín hiệu ra của bộ đếm chỉ xuất hiện (Y = 1) duy nhất trong trường hợp: bộ đếm đang ở trạng thái $M_{đ-1}$ và có tín hiệu vào $X_đ$. Khi đó bộ đếm sẽ chuyển về trạng thái 0.

Trong trường hợp cần hiển thị trạng thái của bộ đếm thì phải dùng thêm mạch giải mã.

2. Phân loại bộ đếm.

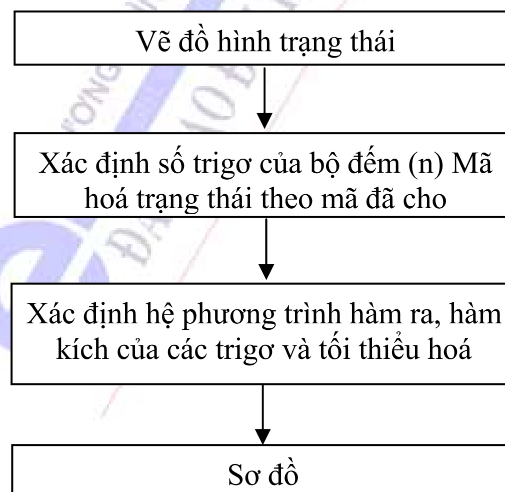
Có nhiều cách phân loại bộ đếm. Hình 5-27 là cách phân loại điển hình của bộ đếm.



Hình 5-27. Sơ đồ phân loại bộ đếm

5.8.1.2. Các bước thiết kế bộ đếm

Hình 5-28 là lưu đồ thiết kế bộ đếm.



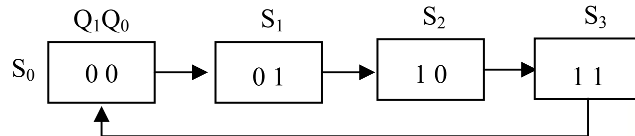
Hình 5-28. Các bước thiết kế bộ đếm

A. Bộ đếm đồng bộ.

A.1. Bộ đếm nhị phân

Thiết kế bộ đếm nhị phân đồng bộ có $M_đ = 4$.

Do $M_đ = 4$ nên lập được đồ hình trạng thái ở hình 5-29.



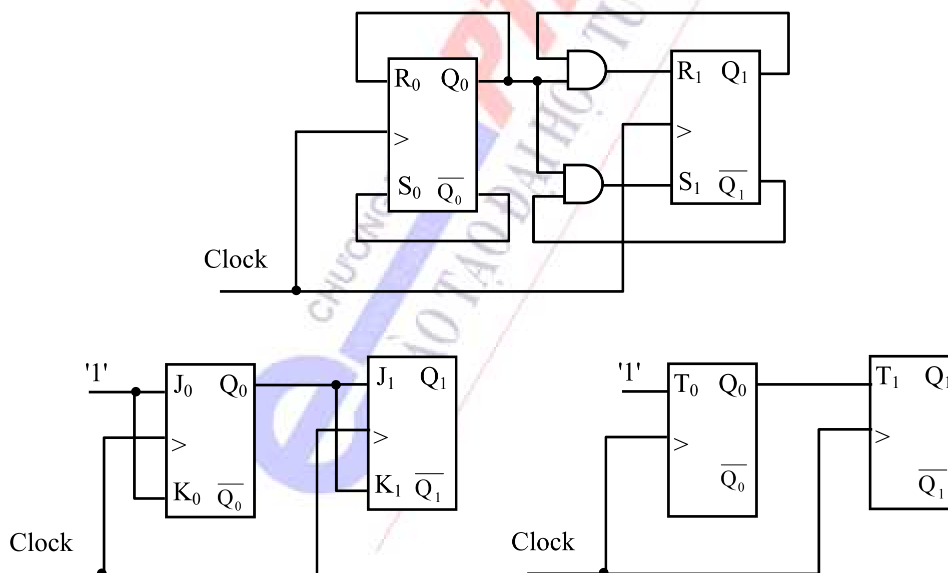
Hình 5-29

Từ đó xác định được số trigơ cần dùng để thiết kế bộ đếm (n = 2) và mã hoá các trạng thái đó. Hai trigơ cần để mã hoá các thái là Q_1 và Q_0. Dùng bảng hàm kích 5-16 để xác định các lối vào kích cho các trigơ.

n		n +1		Trigơ Q_1					Trigơ Q_0				
Q_1	Q_0	Q^k_1	Q^k_0	R_1	S_1	J_1	K_1	T_1	R_0	S_0	J_0	K_0	T_0
0	0	0	1	X	0	0	X	0	0	1	1	X	1
0	1	1	0	0	1	1	X	1	1	0	X	1	1
1	0	1	1	0	X	X	0	0	0	1	1	X	1
1	1	0	0	1	0	X	1	1	1	0	X	1	1

Bảng 5-16

Tối thiểu hoá hàm kích của các trigơ, nhận được kết quả:



Hình 5-30. Bộ đếm Mod 4 dùng trigơ RS, JK, T.

Đối với trigơ Q_0:

$R_0 = Q_0$; $S_0 = \overline{Q_0}$

$J_0 = K_0 = 1$;

$T_0 = 1$;

Đối với trigơ Q_1:

$R_1 = Q_1Q_0$; $S_1 = \overline{Q_1}\,Q_0$

$J_1 = K_1 = Q_0$;

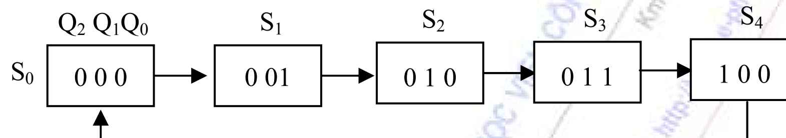
$T_1 = Q_0$;

Sơ đồ mạch điện như ở hình 5-30.

A. 2. Bộ đếm có mod đếm bất kỳ

Thiết kế bộ đếm đồng bộ có $M_đ = 5$.

Do $M_đ = 5$ nên lập được đồ hình trạng thái như hình 5-31.



Hình 5-31. Đồ hình trạng thái của bộ đếm Mod 5

Từ đó xác định được số trigơ cần dùng để thiết kế bộ đếm (n = 3) và mã hoá các trạng thái đó. Ba trigơ cần để mã hoá các thái là Q_2, Q_1 và Q_0. Dùng bảng hàm kích 5-17 để xác định các lối vào kích cho các trigơ.

n			n + 1			Trigơ Q_2		Trigơ Q_1		Trigơ Q_0	
Q_2	Q_1	Q_0	Q^k_2	Q^k_1	Q^k_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X

Bảng 5-17. Bảng hàm kích cho trigơ

Tối thiểu hoá hàm kích của các trigơ, nhận được kết quả:

$J_0 = \overline{Q_2}$; $K_0 = 1$;

$J_1 = K_1 = Q_0$;

$J_2 = Q_1Q_0$; $K_2 = 1$;

Kiểm tra khả năng tự khởi động bằng bảng 5-18.

Nhìn vào bảng trạng thái 5-18, ta thấy các trạng thái dư sau 1 số xung nhịp đều quay trở lại vòng đếm nên ta nói bộ đếm này tự khởi động.

n			n + 1		
Q_2	Q_1	Q_0	Q^k_2	Q^k_1	Q^k_0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	0

Bảng 5-18. Kiểm tra khả năng tự khởi động

Sơ đồ mạch điện ở hình 5-32:

Hình 5-32. Bộ đếm Mod 5 đồng bộ

B. Bộ đếm không đồng bộ.

B. 1. Bộ đếm nhị phân

Các bộ đếm này có sơ đồ rất đơn giản với đặc điểm:

- Chỉ dùng một loại trigơ T hoặc JK. Nếu dùng trigơ T thì lối vào T luôn được nối với mức logic '1', nếu dùng trigơ JK thì J và K được nối với nhau và nối với mức '1'.

- Đầu ra của trigơ trước được nối với lối vào xung nhịp của trigơ sau kế tiếp. Khi đếm tiến thì lấy ở đầu ra Q, khi đếm lùi thì lấy ở đầu ra $\overline{Q}$ (với giả thiết xung Clock tích cực tại sườn âm ↓).

- Tín hiệu vào $X_đ$ luôn được đưa tới lối vào xung nhịp của trigơ có trọng số nhỏ nhất.

Ví dụ đối với bộ đếm nhị phân không đồng bộ $M_đ = 2^n$ dùng các trigơ Q_0, Q_1 …Q_{n-1} với Q_0 là bit có trọng số nhỏ nhất, Q_{n-1} là bit có trọng số lớn nhất, ta có:

- Khi đếm tiến: $C_{Qo} = X$; $C_{Q1} = Q_0 \ldots C_{Qn-1} = C_{Qn-2}$.

- Khi đếm lùi: $C_{Qo} = X$; $C_{Q1} = \overline{Q_0} \ldots C_{Qn-1} = C_{\overline{Qn-2}}$

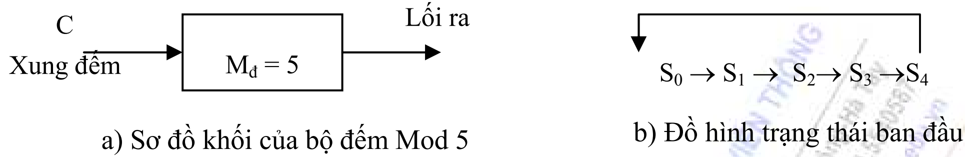
Sơ đồ của bộ đếm nhị phân không đồng bộ 3 bit ($M_đ = 8$ - đếm tiến) dùng trigơ JK được cho ở hình 5-33

Hình 5-33. Bộ đếm nhị phân không đồng bộ 3 bit

B. 2. Bộ đếm có mod đếm bất kỳ.

Ví dụ: Thiết kế bộ đếm $M_đ = 5$ không đồng bộ.

Từ yêu cầu bài toán ta xây dựng sơ đồ khối và đồ hình trạng thái như ở hình 5-34.



a) Sơ đồ khối của bộ đếm Mod 5

b) Đồ hình trạng thái ban đầu

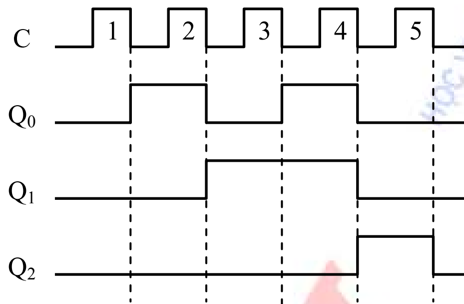
Hình 5-34. Mô hình thiết kế bô đếm

- Chọn lựa mã hoá trạng thái

Có 5 trạng thái nên số trigơ bằng 3, chọn trigơ JK. Chọn mã BCD8421.

$S_0 = 000$; $S_1 = 001$; $S_2 = 010$; $S_3 = 011$; $S_4 = 100$.

- Chọn xung đồng hồ từ giản đồ xung 5-35.



Hình 5-35. Giản đồ xung của bộ đếm Mod 5

$C_1 = \downarrow C$; $C_2 = \downarrow Q_0$; $C_3 = \downarrow C$;

- Tìm hệ phương trình:

Q_2 \ Q_1Q_0	00	01	11	10
0	001	010	100	011
1	000	x	x	x

Q_2 \ Q_1Q_0	00	01	11	10
0	1	0	0	1
1	0	x	x	x

$Q^k_0 = \overline{Q_2}\,\overline{Q_0}$

Q_2 \ Q_1Q_0	00	01	11	10
0	x	1	0	x
1	x	x	x	x

$Q^k_1 = \overline{Q_1}$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	1	0
1	0	x	x	x

$Q^k_2 = \overline{Q_2}\, Q_1 Q_0$

Bảng 5-19. Bảng tính hàm kích của bộ đếm

Sau khi tối thiểu hoá bằng bảng 5-19 ta nhận được hệ phương trình:

$Q^k{}_0 = \overline{Q_2}\,\overline{Q_0}$

$Q^k{}_1 = \overline{Q_1}$

$Q^k{}_2 = \overline{Q_2}\, Q_1 Q_0$

Kiểm tra khả năng tự khởi động bằng bảng 5-20:

n			n + 1		
Q_2	Q_1	Q_0	$Q^k{}_2$	$Q^k{}_1$	$Q^k{}_0$
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	0

Bảng 5-20. Kiểm tra khả năng tự khởi động

Nhìn vào bảng 5-20, ta thấy các trạng thái dư sau 1 số xung nhịp đều quay trở lại vòng đếm nên ta nói bộ đếm này tự khởi động.

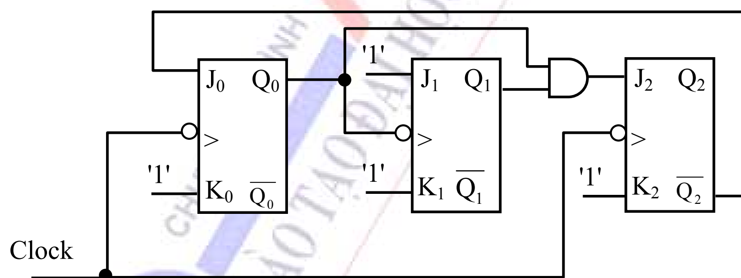
Từ đó ta tìm được phương trình hàm kích:

$J_0 = \overline{Q_2}$; $K_0 = 1$;

$J_1 = K_1 = 1$;

$J_2 = Q_1Q_0$; $K_2 = 1$;

Từ đó ta vẽ được mạch điện của bộ đếm Mod 5 không đồng bộ như hình 5-36.



Hình 5-36. Sơ đồ mạch điện của bộ đếm Mod 5 đồng bộ

5.8.2. Bộ ghi dịch.

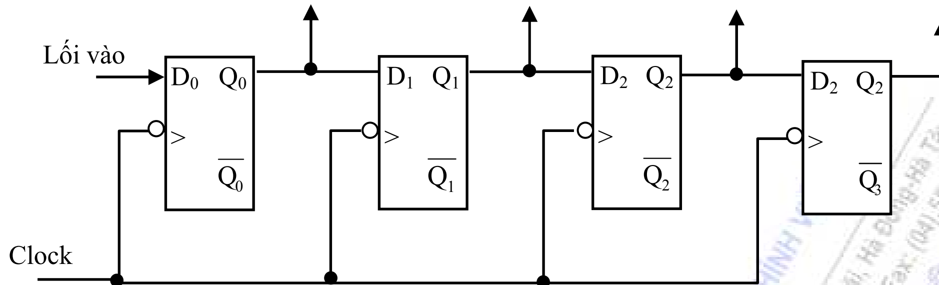
Bộ ghi dịch có khả năng ghi giữ và dịch thông tin.

5.8.2.1. Cấu tạo và phân loại

a) Cấu tạo:

Bộ ghi dịch gồm một dãy các phần tử đơn bit mắc liên tiếp và đóng trên cùng một chip. Các trigơ sử dụng trong bộ ghi dịch thường là trigơ D hoặc các loại trigơ khác mắc theo kiểu D. Để

ghi *n* bit thông tin, người ta sử dụng *n* trigơ, đầu ra của trigơ này mắc tới đầu vào của trigơ kế tiếp. Bộ ghi dịch ghi được *n* bit thông tin được gọi là bộ ghi dịch *n* bit. Hình 5- 37 là sơ đồ của một bộ ghi dịch 4 bit dùng trigơ D



Hình 5-37. Bộ ghi dịch 4 bit dịch phải

Thông tin được nạp vào bộ ghi dịch từng bit một và được đồng bộ với xung nhịp C.

b) Phân loại:

- Phân theo cách đưa tín hiệu vào và lấy tín hiệu ra:

♦ Vào nối tiếp, ra song song: thông tin được đưa vào thanh ghi dịch tuần tự từng bit một, số liệu được đưa ra đồng thời tức là tất cả *n* trigơ của thanh ghi được đọc cùng một lúc.

♦ Vào song song, ra song song: thông tin được đưa vào và lấy ra đồng thời ở *n* trigơ.

♦ Vào nối tiếp, ra nối tiếp: thông tin được đưa vào và lấy ra tuần tự từng bit một.

♦ Vào song song, ra nối tiếp: thông tin được đưa vào đồng thời cả *n* trigơ, lấy ra tuần tự từng bit một dưới sự điều khiển của xung nhịp.

- Phân theo hướng dịch:

♦ Dịch phải, dịch trái, dịch hai hướng, dịch vòng

- Phân theo đầu vào:

♦ Đầu vào đơn: mỗi trigơ trong bộ ghi dịch chỉ sử dụng một đầu vào điều khiển, ví dụ như trigơ D hay các trigơ khác mắc theo kiểu D.

♦ Đầu vào đôi: các trigơ trong bộ ghi dịch sử dụng cả hai đầuvào điều khiển , ví dụ hai lối vào điều khiển của trigơ JK hay trigơ RS.

- Phân theo đầu ra:

♦ Đầu ra đơn: mỗi trigơ trong bộ ghi dịch chỉ có một đầu ra Q_i (hay $\overline{Q_i}$) được đưa ra chân của vi mạch.

♦ Đầu ra đôi: cả hai đầu ra của trigơ Q_i và $\overline{Q_i}$ đều được đưa ra chân của vi mạch.

c) Ứng dụng của bộ ghi dịch

Bộ ghi dịch được sử dụng rộng rãi để nhớ dữ liệu, chuyển dữ liệu từ song song thành nối tiếp và ngược lại. Bộ ghi dịch là thành phần không thể thiếu được trong CPU của các hệ vi xử lý, trong các cổng vào/ra có khả năng lập trình.

Bộ ghi dịch còn được dùng để thiết kế bộ đếm, tạo dãy tín hiệu nhị phân tuần hoàn…

5.8.2.2. Hoạt động cơ bản của bộ ghi dịch

Trong phần này ta giới thiệu bộ ghi dịch 4 bit nạp vào nối tiếp hoặc song song, ra nối tiếp và song song, dịch phải.

Sơ đồ bộ ghi dịch này được trình bày trên hình 5- 37.

Bộ ghi dịch này có thể nạp thông tin vào nối tiếp hoặc song song. Đầu ra nối tiếp được lấy ra ở trigơ cuối cùng, đầu ra song song được lấy ra đồng thời trên cả 4 trigơ. Việc nạp thông tin vào song song được thực hiện bởi một trong hai đầu vào Preset 1 và Preset 2 (đây là 2 lối vào phụ). Trước khi làm việc cần phải xoá tất cả các trigơ về trạng thái '0' nhờ lối vào Clear. Thông tin trong bộ ghi dịch này được dịch phải.

TÓM TẮT

Khác với mạch logic tổ hợp, mạch logic tuần tự có tín hiệu đầu ra phụ thuộc không những tín hiệu đầu vào ở thời điểm xét mà cả vào trạng thái mạch điện sẵn có ở thời điểm đó. Đây là đặc điểm chức năng logic của mạch tuần tự. Để nhớ trạng thái mạch điện, mạch tuần tự phải có phần tử nhớ - đó là các trigơ.

1- Tính chất cơ bản của Trigơ

Trigơ là linh kiện logic cơ bản của mạch số. Trigơ có hai trạng thái ổn định, dưới tác dụng của tín hiệu bên ngoài có thể chuyển đổi từ trạng thái ổn định này sang trạng thái ổn định kia, nếu không có tác dụng tín hiệu bên ngoài thì nó duy trì mãi trạng thái ổn định vốn có. Vì thế, trigơ có thể được dùng làm phần tử nhớ của số nhị phân.

2- Quan hệ giữa chức năng logic và hình thức cấu trúc của trigơ

Chức năng logic và hình thức cấu trúc của trigơ là hai khái niệm khác nhau. Chức năng logic là quan hệ giữa trạng thái tiếp theo của đầu ra với trạng thái hiện tại của đầu ra và các tín hiệu đầu vào. Do chức năng logic khác nhau mà trigơ được phân thành các loại RS, D, T, JK. Còn do hình thức cấu trúc khác nhau mà trigơ lại được phân thành loại trigơ thường và loại trigơ chính phụ.

Một trigơ có chức năng logic xác định có thể thực hiện bằng các hình thức cấu trúc khác nhau. Ví dụ, các trigơ cấu trúc loại chính phụ và loại thường đều có thể thực hiện chức năng của một trigơ khác. Nghĩa là cùng một cấu trúc có thể đảm trách những chức năng khác nhau.

3- Mạch tuần tự cụ thể có rất nhiều chủng loại. Chương này chỉ giới thiệu một số loại mạch tuần tự điển hình: bộ đếm, bộ ghi dịch…Đồng thời với việc nắm vững cấu trúc, nguyên lý công tác và đặc điểm của các mạch tuần tự đó, chúng ta cũng phải nắm vững được đặc điểm chung của mạch tuần tự và phương pháp chung khi phân tích và thiết kế mạch tuần tự.

CÂU HỎI ÔN TẬP CHƯƠNG 5

1. Cho các trigơ cơ bản loại RS, JK, D và T. Loại trigơ nào trong số các loại này có thể thực hiện được mà không cần tín hiệu đồng bộ.

   a. Trigơ RS và trigơ D.

b. Trigơ JK và trigơ T.

c. Trigơ RS và trigơ T.

d. Trigơ JK và trigơ D

2. Trong các loại trigơ sau, trigơ nào còn tồn tại tổ hợp cấm:

a. Trigơ D.

b. Trigơ T

c. Trigơ RS.

d. Trigơ JK.

3. Cần bao nhiêu cổng NAND để thực hiện tạo ra một trigơ RS đồng bộ:

a. 2.

b. 3.

c. 4.

d. 5.

4. Nếu đầu vào D của trigơ thay đổi từ cao đến thấp thì đầu ra

a. thay đổi trạng thái của nó một cách tức thời

b. sẽ thay đổi sau khi có xung nhịp clock ở đầu vào .

c. sẽ thay đổi sau khi có 2 xung nhịp clock ở đầu vào .

d. sẽ không thay khi có xung nhịp tiếp theo.

5. Một trigơ JK được ở chế độ lật. Nếu tần số Clock của nó là 1000 hz thì tần số tại lối ra là:

a. 2000 hz.

b. 1000 hz.

c. 100 hz.

d. 500 hz.

6. Mô hình Mealy là mô hình:

a. có hàm ra phụ thuộc vào tín hiệu vào và trạng thái trong của mạch.

b. có hàm ra phụ thuộc vào tín hiệu vào.

c. có hàm ra phụ thuộc vào trạng thái trong của mạch.

d. không có phương án nào đúng.

7. Mô hình Moore là mô hình:

a. có hàm ra phụ thuộc vào tín hiệu vào và trạng thái trong của mạch.

b. có hàm ra phụ thuộc vào tín hiệu vào.

c. có hàm ra phụ thuộc vào trạng thái trong của mạch.

d. không có phương án nào đúng.

8. Các phương pháp mô tả mạch tuần tự:

a. Bảng chuyển đổi trạng thái.

b. Bảng tín hiệu ra.

c. Đồ hình trạng thái.

d. Cả ba phương án trên đều đúng.

9. Các phần tử nhớ của bộ ghi dịch là:

a. Trigơ D.

b. Trigơ RS.

c. Trigơ JK.

d. Bất kỳ loại trigơ nào nhưng phải đưa về dạng trigơ D.

10. Cần bao nhiêu trigơ để thực hiện tạo ra một bộ ghi dịch 4 bit:

a. 2.

b. 3.

c. 4.

d. 5.

11. Bằng cách nào tạo ra được một Trigơ Chính - phụ (MS):

a. Từ hai trigơ cùng loại đồng bộ.

b. Từ hai trigơ cùng loại.

c. Từ ba trigơ cùng loại.

d. Từ 4 trigơ cùng loại.

12. Bộ đếm mã Johnson là:

a. Bộ đếm vòng.

b. Bộ đếm vòng xoắn.

c. Bộ đếm nhị phân.

d. Cả ba phương án trên đều đúng.

13. Một bộ đếm nhị phân 4 bit thì tần số tại lối ra của bit có trọng số lớn nhất so với tần số xung nhịp:

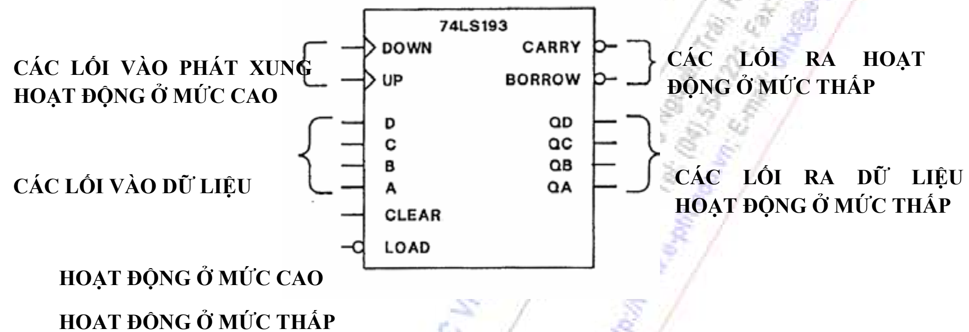
a. nhỏ hơn 2 lần.

b. nhỏ hơn 4 lần.

c. nhỏ hơn 8 lần.

d. nhỏ hơn 16 lần.

14. Trên bộ đếm đồng bộ, các lối vào Clock
    a. phải được nối với tầng LSB của bộ đếm.
    b. phải được nối với tầng MSB của bộ đếm.
    c. là chung cho mỗi tầng của bộ đếm.
    d. phải là dạng xung được phát theo kiểu đơn bước.



Hình 1

15. Với IC xuất hiện trên hình 1, chân CLEAR
    a. xoá tất cả 6 lối ra của IC.
    b. lập tất cả 6 lối ra của IC.
    c. Chỉ xoá các lối ra từ QD đến QA.
    d. Chỉ xoá các lối ra CARRY và BORROW.

16. Nếu các lối vào của LS 193 có giá trị là 1010, thì các lối ra của bộ đếm sẽ là:
    a. hiển thị giá trị 1010 sau khi chức năng LOAD được kích hoạt.
    b. hiển thị giá trị 0101 là giá trị đảo của 1010 sau khi chức năng LOAD được kích hoạt.
    c. hiển thị giá trị 1010 sau một xung clock.
    d. sẽ tăng lên nhưng không thể giảm xuống.

17. Các lối ra CARRY và BORROW của bộ đếm LS 193:
    a. bình thường ở mức thấp và sẽ phát ra một xung hoạt động ở mức cao.
    b. có thể được đưa lên mức cao bằng cách kích hoạt chức năng LOAD.
    c. có thể được đưa xuống mức thấp bằng cách kích hoạt chức năng CLEAR.
    d. bình thường ở mức cao và sẽ phát ra một xung hoạt động ở mức thấp.

18. Trên bộ đếm LS 193, bộ đếm thực hiện đếm tiến:
    a. nếu chân DOWN được cấp xung và chân UP nối lên V_{CC}.

b. nếu chân UP được cấp xung và chân DOWN nối lên V_{CC}.

c. chân UP và DOWN được cấp xung đồng thời.

d. chân UP và DOWN đều được nối lên V_{CC}.

19. Trên bộ đếm LS 193, bộ đếm thực hiện đếm lùi:

a. nếu chân DOWN được cấp xung và chân UP nối lên V_{CC}.

b. nếu chân UP được cấp xung và chân DOWN nối lên V_{CC}.

c. chân UP và DOWN được cấp xung đồng thời.

d. chân UP và DOWN đều được nối lên V_{CC}.

20. Một bộ đếm không đồng bộ 5 bit thì cung cấp hệ số chia tần hay hệ số chia số đếm là bao nhiêu :

a. 32.

b. 16.

c. 8.

d. Không có trường hợp nào ở trên.

21. Với bộ đếm không đồng bộ, qua mỗi trigơ thì lối ra của nó chia tần số đầu vào ra làm :

a. 4.

b. 2.

c. 10.

d. 16.

22. Tần số đầu vào của một bộ đếm không đồng bộ 4 bit là 100KHz. Vậy tần số tại đầu ra tại lối ra có trọng số lớn nhất (MSB) là bao nhiêu ?

a. 100 KHz.

b. 50 KHz.

c. 12,5 KHz.

d. 6, 25 KHz.

23. Khi tần số xung nhịp của bộ đếm không đồng bộ tăng thì :

a. Các đầu vào xoá (CLEAR) và lập (SET) không điều khiển tất cả các trigơ của bộ đếm.

b. Chức năng của các đầu vào xoá (CLEAR) và lập (SET) không bị ảnh hưởng gì.

c. Tăng khả năng đếm lớn nhất của nó.

d. Giảm khả năng đếm lớn nhất của nó.

24. Một xung clock vào :

a. Cho phép một bộ đếm không đồng bộ chạy trong chế độ không đồng bộ.

b. Xác định số đếm lớn nhất của bộ đếm không đồng bộ.

c. Thay đổi lần lượt các chế độ hoạt động của bộ đếm không đồng bộ.

d. Chuyển một bộ đếm không đồng bộ thành một bộ đếm nối tiếp.

25. Khi phát xung vào bộ đếm không đồng bộ thì xung clock là :

a. Tín hiệu điều khiển tất cả các đầu vào.

b. Tín hiệu điều khiển tầng LSB của bộ đếm.

c. Tín hiệu điều khiển tầng MSB của bộ đếm.

d. Trạng thái tĩnh.

26. Khi chân CLEAR (xoá) của bộ đếm không đồng bộ được đưa xuống mức thấp thì bộ đếm :

a. Không tiếp nhận xung xoá bởi vì xung CLOCK chạy tự do.

b. Tiếp nhận xung xoá, lúc này tất cả các đầu ra không đảo được đặt cố định ở mức thấp.

c. Tiếp nhận xung xoá, lúc này tất cả các đầu ra không đảo được đặt tạm thời ở mức thấp.

d. Dao động giữa giá trị đếm lớn nhất và giá trị nhỏ nhất.

27. Khi chân SET (lập) của bộ đếm không đồng bộ được đưa xuống mức thấp thì bộ đếm:

a. Không tiếp nhận xung lập bởi vì xung CLOCK chạy tự do.

b. Tiếp nhận xung lập, lúc này tất cả các đầu ra không đảo được đặt cố định ở mức cao.

c. Tiếp nhận xung lập, lúc này tất cả các đầu ra không đảo được đặt tạm thời ở mức cao.

d. Dao động giữa giá trị đếm lớn nhất và giá trị nhỏ nhất.

28. Một bộ đếm không đồng bộ được coi như là một bộ đếm nối tiếp là bởi vì :

a. Tất cả các đầu ra thay đổi đồng thời.

b. Một tín hiệu xung nhịp điều khiển tất cả các trigơ.

c. Tất cả các đầu ra là đảo.

d. Các trigơ trong bộ đếm hoạt động theo phương pháp chuỗi cánh hoa (daisy-chaind). (Điều này có nghĩa là lối ra của trigơ trước sẽ điều khiển lối vào của trigơ sau).

29. Hệ số chia tần số cho một bộ đếm không đồng bộ 4 bit là :

a. 1, 2, 4 và 8.

b. 1, 2, 4 và 16.

c. 2, 4, 8 và 16.

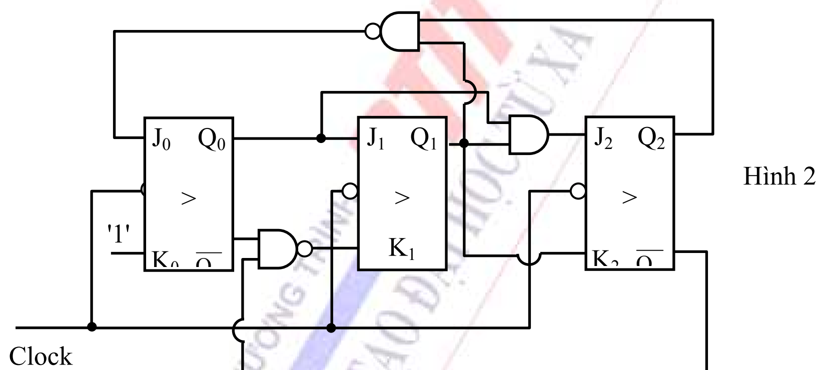
d. Tất cả các trường hợp trên, phụ thuộc vào tần số xung clock.

30. Nếu một bộ đếm không đồng bộ 4 bit có các lối ra đảo thì chúng
    - a. Đếm từ 15 → 0.
    - b. Đếm từ 0 → 15.
    - c. Luôn là 0.
    - d. Luôn là 15.

31. Cần bao nhiêu chu kỳ xung clock đầu vào để phát ra một chu kỳ hoàn chỉnh tại lối ra có trọng số lớn nhất (MSB) của bộ đếm không đồng bộ 4 bit.
    - a. 32.
    - b. 16.
    - c. 8.
    - d. Không có trường hợp nào ở trên.

32. Các Trigơ JK sử dụng trong bộ đếm không đồng bộ được xây dựng bằng cách:
    - a. Nối lối vào J và K với V_{CC} và vô hiệu hoá các lối vào CLR (xoá) và PR (lập).
    - b. Cấu trúc mạch Trigơ JK giống như một mạch Trigơ T.
    - c. Nối tất cả các lối vào J, K, CLR và PR với V_{CC}.
    - d. Sử dụng bất kỳ cấu trúc nào ở trên.

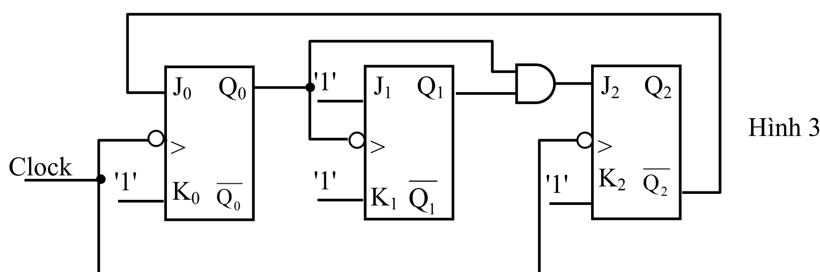
33. Cho bộ đếm hình 2. Cho biết đây là bộ đếm Mod mấy?



Hình 2

- a. Mod 5.
- b. Mod 6.
- c. Mod 7.
- d. Mod 8.

34. Cho bộ đếm hình 3. Cho biết đây là bộ đếm Mod mấy?



Hình 3

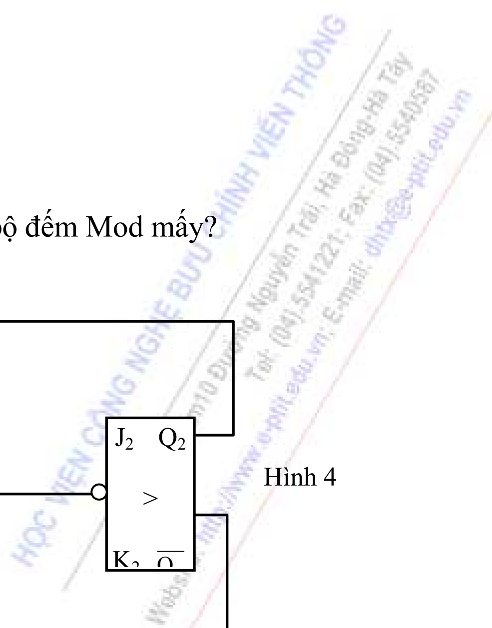
a. Mod 5.

b. Mod 6.

c. Mod 7.

d. Mod 8.

35. Cho bộ đếm hình 4. Cho biết đây là bộ đếm Mod mấy?



Hình 4

a. Mod 5.

b. Mod 6.

c. Mod 7.

d. Mod 8.

36. Thiết kế bộ đếm Mod 9 đồng bộ.

37. Thiết kế bộ đếm Mod 9 không đồng bộ.

38. Bộ ghi dịch của bạn được reset. Sau 4 sườn dương của xung clock tất cả 4 lối ra đều ở mức cao. Kết luận của bạn về các lối vào dữ liệu là:

a. được đặt ở mức thấp.

b. Lần lượt thay đổi giữa hai trạng thái cao và thấp.

c. Lần lượt thay đổi giữa hai trạng thái thấp và cao.

d. được đặt ở mức cao.

39. Nếu mạch của bạn được thiết kế để dịch trái dữ liệu vào nối tiếp, sau đó luồng bit dữ liệu chuyển động từ:

a. Trái qua phải.

b. Từ phải qua trái.

c. Một trong hai trường hợp trên.

d. Không có trường hợp nào ở trên.

40. Nếu mạch của bạn được định hình để dịch phải dữ liệu vào nối tiếp, sau đó luồng bit dữ liệu chuyển động từ:

a. Trái qua phải.

b. Từ phải qua trái.

c. Một trong hai trường hợp trên.

d. Không có trường hợp nào ở trên.

CHƯƠNG 6: MẠCH PHÁT XUNG VÀ TẠO DẠNG XUNG

GIỚI THIỆU

Hầu hết các hệ thống kỹ thuật số đều yêu cầu một vài loại dạng sóng định thời, ví dụ một nguồn xung của bộ dao động cần thiết cho tất cả các hệ thống tuần tự định thời. Trong các hệ thống kỹ thuật số, một dạng sóng xung vuông thường được sử dụng nhất. Sự tạo ra các dạng sóng xung vuông được gọi là bộ đa hài.

Có ba loại bộ đa hài:

- Bộ dao động đa hài (chạy tự do).
- Bộ đa hài đơn ổn (một nhịp).
- Bộ đa hài hai trạng thái ổn định (trigơ).

Một bộ dao động đa hài chỉ là một bộ dao động để tạo ra dạng xung. Nó có hai trạng thái chuẩn mà không yêu cầu sự kích hoạt từ bên ngoài. Bộ này thường được dùng làm xung điều khiển cho các mạch tuần tự.

Một bộ đa hài đơn ổn chỉ có một trạng thái ổn định, tức là trong điều kiện trạng thái ổn định thì đầu ra của nó cố định. Đầu ra này ở trạng thái LOW hoặc ở trạng thái HIGH. Mạch này cần một xung kích khởi từ bên ngoài để cho mạch chuyển sang trạng thái khác. Mạch này vẫn giữ nguyên trạng thái cũ trong một khoảng thời gian, khoảng thời gian này phụ thuộc vào các thành phần được dùng trong mạch. Trạng thái của mạch này được xem là trạng thái ổn định bởi vì nó phục hồi trở về trạng thái ổn định mà không cần bất kỳ xung kích hoạt nào từ bên ngoài. Độ rộng của xung kích khởi rất nhỏ, độ rộng của xung đầu ra chỉ phụ thuộc vào khoảng thời gian mà mạch giữ lại ở trạng thái ổn định. Mạch này được gọi là mạch một nhịp (one-shot) bởi vì một xung kích khởi chỉ tạo được một xung nhưng độ rộng xung lại khác. Mạch này rất hữu dụng bởi vì nó có thể tạo ra một xung tương đối dài (hàng chục mili giây) từ một xung hẹp, do đó nó còn được gọi là bộ giảm xung (pulse stretcher).

Ví dụ, một bộ vi xử lý có thể phát tín hiệu cho một thiết bị bên ngoài để in một nội dung nào đó bằng cách truyền qua một xung. Thiết bị đầu ra nói chung có tốc độ chậm hơn bộ vi xử lý, do đó nó yêu cầu một xung tín hiệu trong một khoảng thời gian lâu hơn. Điều này đạt được bằng một mạch giao tiếp có chứa bộ đa hài đơn ổn.

Một mạch đa hài trong đó cả hai trạng thái đều ổn định thì được gọi là mạch đa hài hai trạng thái ổn định hay trigơ. Mạch này thực hiện việc chuyển tiếp từ một trạng thái ổn định này sang một trạng thái ổn định khác chỉ lúc xung kích khởi được áp vào. Mạch này thường được dùng làm các thành phần trong bộ nhớ trong các hệ thống kỹ thuật số và đã được thảo luận ở chương 5.

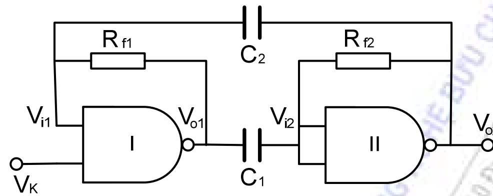
Chương này tập trung vào sơ đồ, nguyên tắc hoạt động, ứng dụng của các mạch dao động đa hài, mạch dao động đa hài đợi, trigơ Schmitt dựa trên các cổng TTL, CMOS và IC định thời 555. Sau chương này độc giả có thể tự thiết kế các mạch dao động theo các yêu cầu cơ bản cho các ứng dụng khác nhau.

NỘI DUNG

6.1. MẠCH PHÁT XUNG

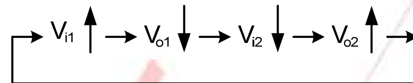
6.1.1. Mạch dao động đa hài cơ bản cổng NAND TTL

Cổng NAND khi làm việc trong vùng chuyển tiếp có thể khuếch đại mạnh tín hiệu đầu vào. Nếu 2 cổng NAND được ghép điện dung thành mạch vòng như hình 6-1 ta được bộ dao động đa hài.V_K là đầu vào điều khiển, khi ở mức cao mạch phát xung, và khi ở mức thấp mạch ngừng phát.

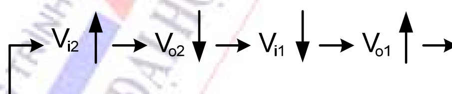


Hình 6-1. Bộ dao động đa hài cấu trúc bằng cổng NAND

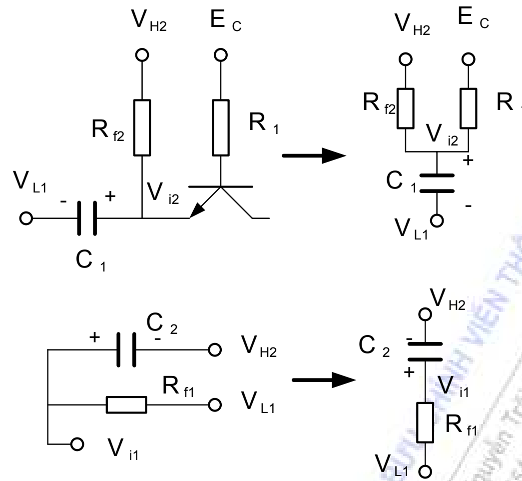
Nếu các cổng I và II thiết lập điểm công tác tĩnh trong vùng chuyển tiếp và $V_K = 1$, thì mạch sẽ phát xung khi được nối nguồn. Nguyên tắc làm việc của mạch như sau: Giả sử do tác động của nhiễu làm cho V_{i1} tăng một chút, lập tức xuất hiện quá trình phản hồi dương sau:



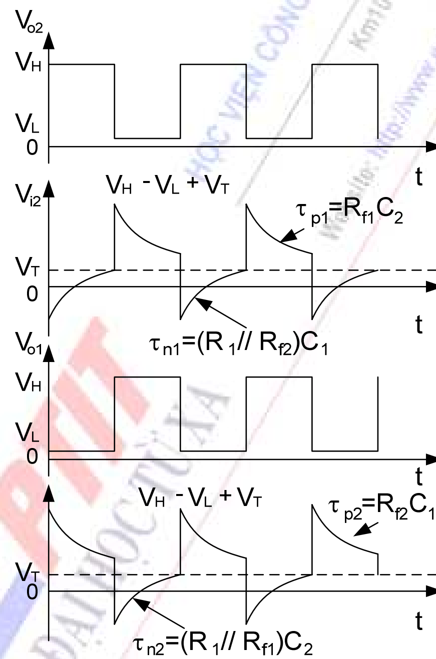
Khi đó, cổng I nhanh chóng trở thành thông bão hoà, cổng II nhanh chóng ngắt, mạch bước vào trạng thái tạm ổn định. Lúc này, C_1 nạp điện và C_2 phóng điện theo mạch đơn giản hoá được thể hiện trong hình 6-2. C_1 nạp đến khi V_{i2} tăng đến ngưỡng thông V_T, trong mạch xuất hiện quá trình phản hồi dương như sau:



Kết quả quá trình này là: cổng I nhanh chóng ngắt còn cổng II thông bão hoà, mạch điện bước vào trang thái tạm ổn định mới. Lúc này C_2 nạp điện còn C_1 phóng cho đến khi V_{i1} bằng ngưỡng thông V_T làm xuất hiện quá trình phản hồi dương đưa mạch về trạng thái ổn định ban đầu. Mạch không ngừng dao động, khi bỏ qua điện trở đầu ra của các cổng NAND, dựa vào hình 6-2 giản đồ xung của mạch được thể hiện trên hình 6-3.



Hình 6-2. Mạch vòng nạp phóng điện của tụ C1, C2



Hình 6-3. Dạng sóng gần đúng của điện áp tại các điểm trên mạch bộ dao động đa hài.

Vì thời gian nạp điện nhanh hơn thời gian phóng, nên thời gian duy trì trạng thái ổn định tạm thời phụ thuộc vào thời gian nạp điện của hai tu điện C_1 và C_2. Từ hình 6-2 ta có thời gian nạp điện của tu C_1 là $\tau_1 = (R_{f2} // R_1)\ C_1$, thời gian để V_{i2} nạp điện đến V_T là:

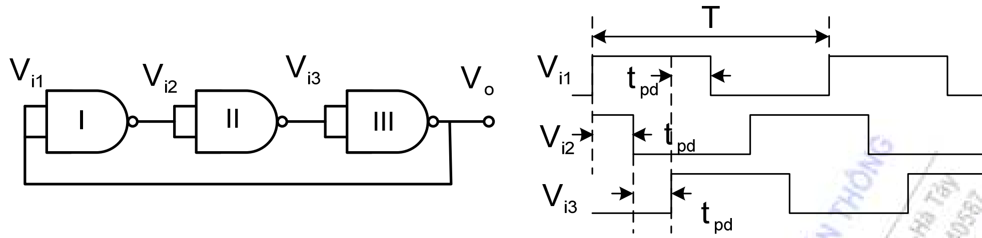
$$t_{M2} = \left(R_{f2} // R_1\right)C_1 \ln \frac{2V_{OH} - \left(V_T + V_{OL}\right)}{V_{OH} - V_T}$$

Nếu $R_{f1}=R_{f2}=R_f$, $C_1=C_2=C$, $V_{OH}=3$ V, $V_{OL}=0,35$ V, $V_T = 1,4$ V thì ta có:

$$T \approx 2\left(R_f // R_1\right)C$$

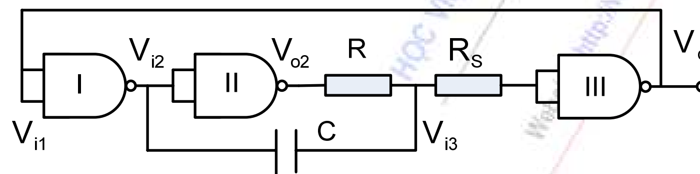
T là chu kỳ của tín hiệu đa hài lối ra.

6.1.2. Mạch dao động đa hài vòng RC



Hình 6-4. Bộ dao động vòng và dạng sóng

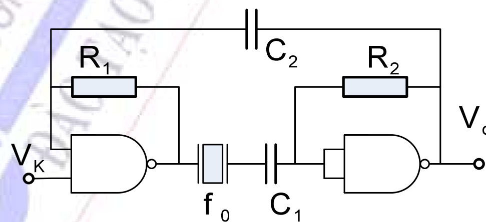
Bộ dao động vòng có cấu trúc gồm 3 cổng NAND mắc nối tiếp như hình 6-4. Phản hồi dương từ V_o đến V_{i1} làm cho mạch này không có trạng thái ổn định. Tần số của tín hiệu lối ra phụ thuộc vào thời gian trễ của cổng NAND, và không thể điều chỉnh được tần số này. Tần số của mạch phát sẽ điều chỉnh được khi một mạch trễ RC được mắc thêm vào mạch như hình 6-5. Tần số dao động của mạch điều chỉnh được thông qua giá trị của tụ điện C và điện trở R.



Hình 6-5. Bộ dao động đa hài có mạch RC

6.1.3. Mạch dao động đa hài thạch anh

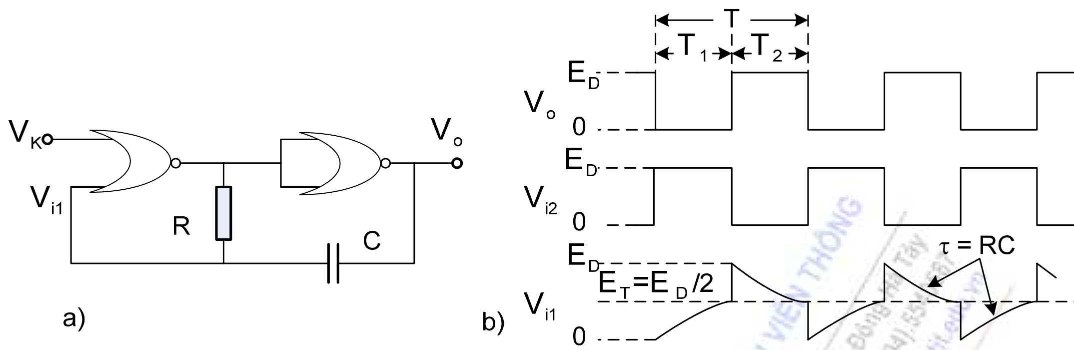
Để có các tín hiệu đồng hồ có tần số chính xác và có độ ổn định cao, các mạch đa hài trình bày trên đây không đáp ứng được. Tinh thể thạch anh thường được sử dụng trong các trường hợp này. Thạch anh có tính ổn định tần số tốt, hệ số phẩm chất rất cao dẫn đến tính chọn lọc tần số rất cao. Hình 6-6 là một mạch dao động đa hài điển hình sử dụng tinh thể thạch anh. Tần số của mạch dao động chỉ phụ thuộc vào tinh thể thạch anh mà không phụ thuộc vào giá trị các tụ điện và điện trở trong mạch.



Hình 6-6. Mạch dao động đa hài thạch anh

6.1.4. Mạch dao động đa hài CMOS

Hình 6-7a là mạch dao động đa hài cơ bản sử dụng hai cổng NOR CMOS và các linh kiện định thời trở và tụ. Giản đồ xung của mạch được thể hiện trên hình 6-7b. Chu kỳ dao động của mạch được tính gần đúng như sau:

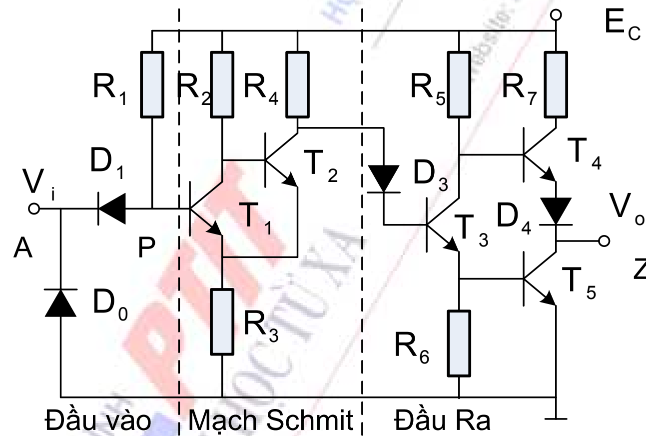


Hình 6-7. Bộ dao động đa hài dùng cổng NOR CMOS và giản đồ xung

$$T = T_1 + T_2 = RC\ln\left(\frac{E_D}{E_D - V_T} + \frac{E_D}{V_T}\right)$$

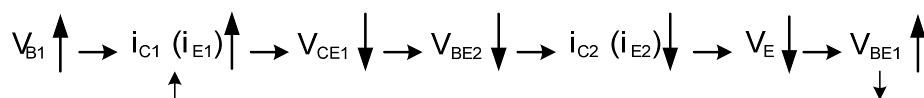
Nếu giả thiết $V_T = E_D/2$ thì $T_1 = T_2$, khi đó $T = RC\ln4 \approx 1{,}4RC$.

6.2. TRIGƠ SCHMIT

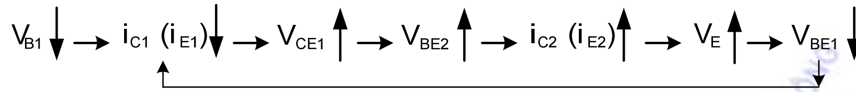


Hình 6-8. Sơ đồ nguyên lý của trigơ Schmit

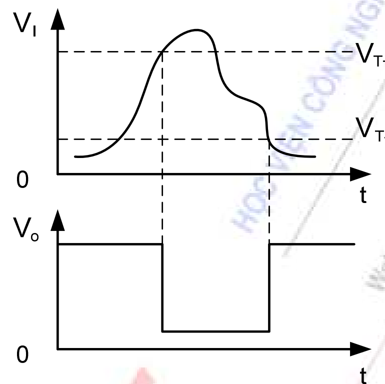
Hình 6-8 là sơ đồ nguyên lý của trigơ schmitt, hay còn được gọi là bộ đảo pha trigơ schmit. Nó gồm 3 phần: mạch đầu vào, mạch schmit và tầng công suất lối ra. Nguyên tắc làm việc của mạch như sau: Nếu V_{B1} ở mức thấp thì T_1 ngắt, T_2 thông bão hoà và ngược lại nếu V_{B1} ở mức cao thì T_1 thông bão hoà, T_2 ngắt. Khi V_{B1} tăng từ mức thấp lên mức cao đến trị số $V_{BE1} = V_{B1} - I_LR_3 = 0{,}5$ V thì T_1 bắt đầu chuyển từ trạng thái ngắt vào trạng thái khuếch đại. Do V_{B1} tiếp tục tăng nên $V_{CE1} = V_{BE2}$ giảm xuống. Sau khi T_2 rời khỏi trạng thái bão hoà mà V_{B1} tiếp tục tăng thì xảy ra quá trình phản hồi dương sau:



Nhờ phản hồi dương mạch điện nhanh chóng chuyển sang trạng thái T_1 thông bão hoà, T_2 ngắt. Nếu V_{B1} sau khi tăng đến cực đại thì bắt đầu giảm; khi V_{B1} giảm đến mức làm T_1 ra khỏi vùng bão hoà, T_2 ra khỏi vùng ngắt thì mạch điện lại xảy ra quá trình phản hồi dương sau:



Kết quả mạch điện nhanh chóng lật sang trạng thái T_1 ngắt, T_2 thông bão hoà. Chúng ta gọi giá trị điện áp đầu vào V_I trong quá trình tăng lên của nó đạt đến ngưỡng làm lật mạch schmit để đầu ra từ mức cao xuống mức thấp là ngưỡng trên V_{T+} và giá trị ngược lại là ngưỡng dưới của trigơ schmit V_{T-}(hình 6- 9). Hiệu điện áp tương ứng với ngưỡng trên và ngưỡng dưới được gọi là đọ chênh lệch điện áp chuyển mạch $\Delta V = V_{T+} - V_{T-}$.



Hình 6-9. Dạng sóng đầu vào V_I và đầu ra V_O của trigơ schmit

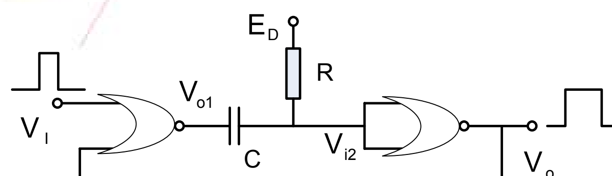
Trigơ schmit thực chất là một bộ so sánh hai ngưỡng nên nó được dùng ứng dụng khác nhau như: Các mạch dao động, các mạch so sánh, lọc nhiễu v.v..

6.3. MẠCH ĐA HÀI ĐỢI

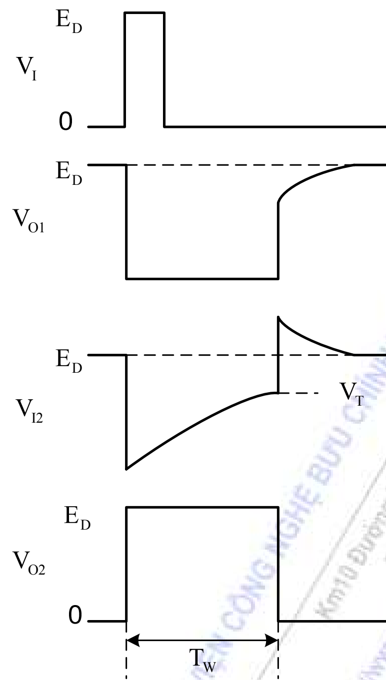
Mạch đa hài đợi có một trạng thái ổn định và một trạng thái tạm ổn định. Khi có tác dụng của xung ngoài, mạch có thể chuyển đổi từ trạng thái ổn định sang trạng thái tạm ổn định. Sau khi duy trì một thời gian, mạch sẽ tự động quay lại trạng thái ổn định. Thời gian tạm ổn định phụ thuộc vào các thông số của mạch mà không phụ thuộc vào xung kích. Mạch đa hài được ứng dụng trong các mach định thời, tạo dạng xung, trễ v.v..

6.3.1. Mạch đa hài đợi CMOS

1. Mạch đa hài đợi kiểu vi phân



Hình 6-10. Đa hài đợi kiểu vi phân dùng cổng NOR CMOS



Hình 6-11. Dạng song của mạch đa hài đợi kiểu vi phân

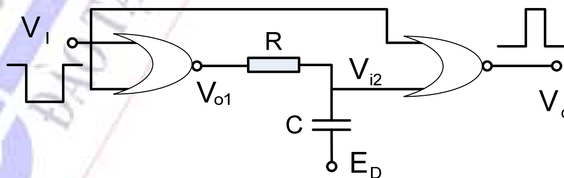
Hình 6-10 là sơ đồ nguyên lý của mach đa hài đợi kiểu vi phân. Tại trạng thái ổn định, V_I=0 thì V_{O1}=E_D, V_{I2}=E_D, V_{O2}=0. Khi có một xung kích thích lối vào làm cho cổng 1 nhanh chóng cấm và lối ra bằng 0, xem giản đồ 6-11. Mạch điện RC sẽ nạp điện cho tụ điện C. Trong quá trình nạp, điện áp V_{I2} tăng dần đến ngưỡng V_T và làm cổng 2 đóng, điện áp V_{O2}=0. Khi đó, cổng 1 nhanh chóng chuyển về trạng thái cấm và làm cho mạch đa hài đợi trở về trạng thái ổn định.

Độ rộng xung tại đầu ra của mạch được xác định bằng công thức sau:

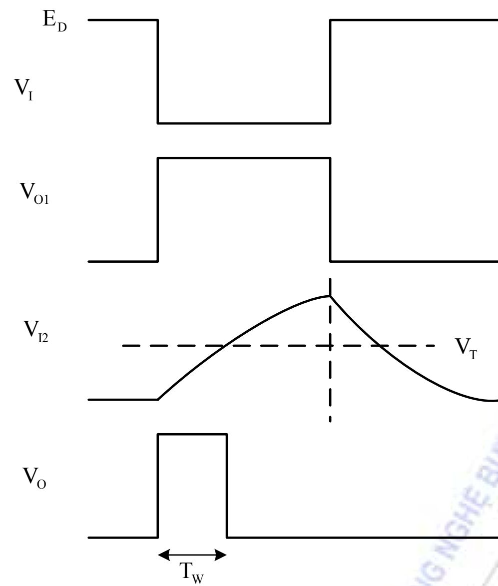
$$T_W = (R + R_0)C * \ln \frac{E_D}{E_D - V_T}$$

trong đó R_0 là điện trở đầu ra của cổng 1, nếu V_T=E_D/2 thì $T_W = 0,7(R + R_0)C$

2. Mạch đa hài đợi kiểu tích phân



Hình 6-12. Đa hài đợi kiểu tích phân dùng cổng NOR CMOS



Hình 6-13. Dạng sóng của mạch đa hài đợi kiểu tích phân

Hình 6-12 biểu diễn sơ đồ nguyên lý của mạch đa hài đợi kiểu tích phân. Tại trạng thái ổn định, V_I=1 thì V_{O1}=0, V_{I2}=0, V_{O2}=0. Khi lối vào V_I chuyển từ 1 xuống 0 lối ra V_{O2} nhảy từ trạng thái 0 lên 1 và đồng thời mạch RC bắt đầu tích điện cho tụ điện C, khi điện áp $V_{I2} = V_T$ điện áp lối ra V_{O2} chuyển xuống trạng thái 0. Sau khi khi hết xung lối vào tụ điện phóng điện thông qua trở R và mạch trở về trạng thái ổn định.

Độ rộng xung lối ra của mạch đa hài đợi được tính theo công thức:

$$T_W = (R + R_0)C * \ln \frac{E_D}{E_D - V_T}$$

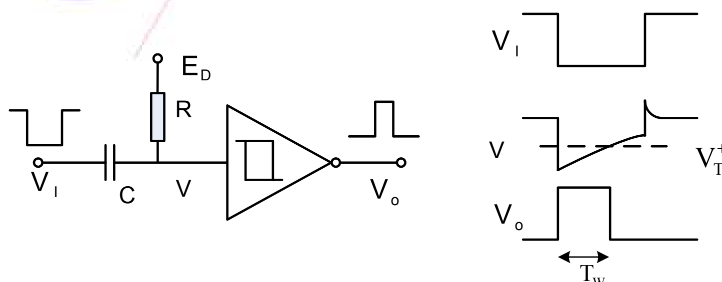
trong đó R_0 là điện trở đầu ra của cổng 1, nếu V_T=E_D/2 thì $T_W = 0,7(R + R_0)C$

3. Mạch đa hài đợi dùng trigơ Schmitt

Dựa vào đặc tính so sánh của trigơ Schmitt, mạch nguyên lý chỉ ra trên hình 6-14 là bộ đa hài đợi. Độ rộng xung lối ra phụ thuộc vào ngưỡng trên của trigơ Schmitt và giá trị của tụ điện C và điện trở R theo công thức sau:

$$T_W = RC * \ln \frac{E_D}{E_D - V_T^+}$$

nếu V_T=E_D/2 thì $T_W = 0,7RC$



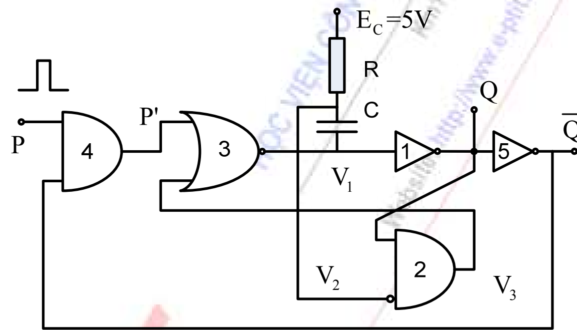
Hình 6-14. Sơ đồ nguyên lý và giản đồ thời gian của mạch đa hài dùng trigơ Schmitt

6.3.2. Mạch đa hài đợi TTL

Hình 6-15 là sơ đồ nguyên lý mạch đa hài đợi họ TTL, trong đó các cổng 1, 2, 3 cấu trúc lên mạch flip-flop, cổng 4,5 là mạch tạo dạng xung. Các cổng này thuộc họ TTL nên có mức logic 1 là 3,6 V và logic 0 là 0,3 V. Đầu vào V_2 biểu thị sử dụng mạch đảo. Mạch đảo này thông bão hoà thì $V_2 \sim 0{,}7$ V, còn ngưỡng thông của nó cỡ 0,6 V.

Tại trạng thái ổn định P = P' = 0. Mạch đảo đầu vào V_2 là bộ khuếch đại transistor emitter chung ở trạng thái bão hoà và khi đó $V_2 = 0{,}7$ V, $V_3 = 0$, $V_1 = 1$, Q = 0, $\overline{Q} = 1$.

Khi có xung dương tác động ở đầu vào thì P = 1, P' = 1, $V_1 = 0$, Q = 1, $\overline{Q} = 0$, mạch ở trạng thái tạm ổn định. Do $\overline{Q} = 0$ khoá cổng 4, nên sau khi bị kích thích bởi sườn dương xung P thì mạch bị cách ly khỏi xung P.



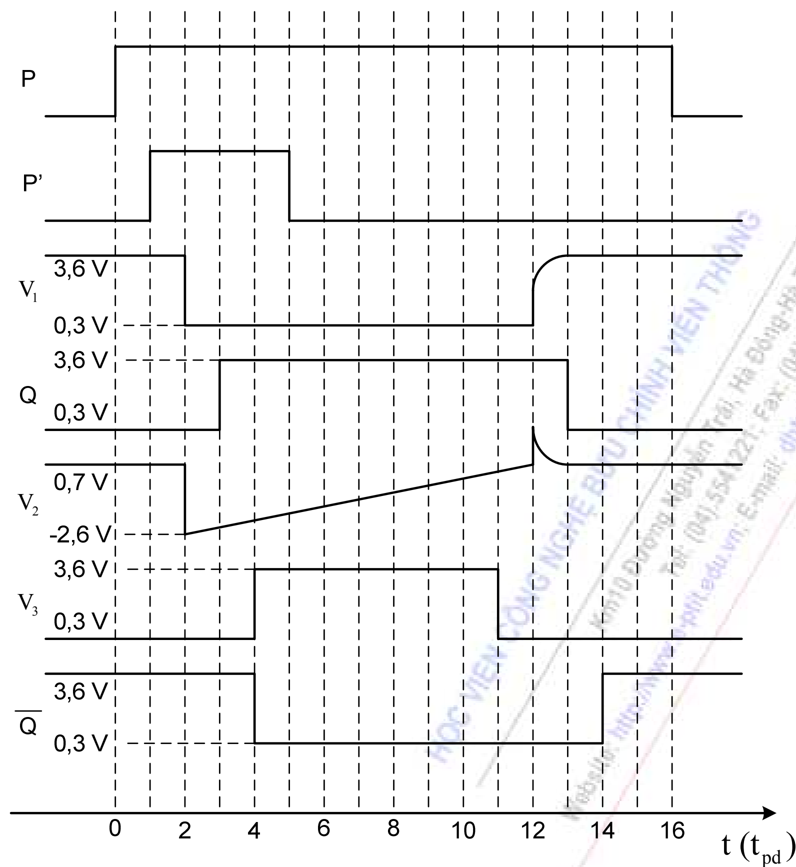
Hình 6-15. Sơ đồ nguyên lý mạch đa hài đợi họ TTL

Vì điện áp trên tụ C không tăng đột biến nên khi V_1 từ mức cao 3,6 V đột biến xuống 0,3 V thì V_2 từ mức 0,7 V đột biến xuống -2,6 V. Bắt đầu quá trình nạp điện của tụ điện C. V_2 tăng dần lên. Khi V_2 Tăng lên đến ngưỡng thông 0,6 V thì sinh ra quá trình phản hồi dương sau:

$$V_2 \uparrow \rightarrow V_3 \downarrow \rightarrow V_1 \uparrow \rightarrow Q \downarrow$$

Quá trình này làm mạch nhanh chóng trở về trạng thái ổn định ban đầu $V_3 = 0$, $V_1 = 1$, Q = 0, $\overline{Q} = 1$. Tiếp đó tụ điện C phóng điện, V_2 dần dần hồi phục về 0,7 V. Hình 6-16 chỉ ra giản đồ xung của mạch đa hài đợi họ TTL với giả thiết thời gian trễ truyền đạt của các cổng và bộ đảo pha đều bằng t_{pd}.

Độ rộng xung ra được tính theo công thức $T_W = 0{,}7RC$. Mạch dao động đa hài đợi được thiết kế sẵn trong một số họ IC TTL như 74LS121, 74LS123 … bằng cách thay đổi các giá trị tụ và trở mắc ngoài sẽ cho các xung lối ra mong muốn.



Hình 6-16. Giản đồ xung của mạch dao động đa hài đợi TTL với giả thiết độ trễ của các cổng là t_{pd}.

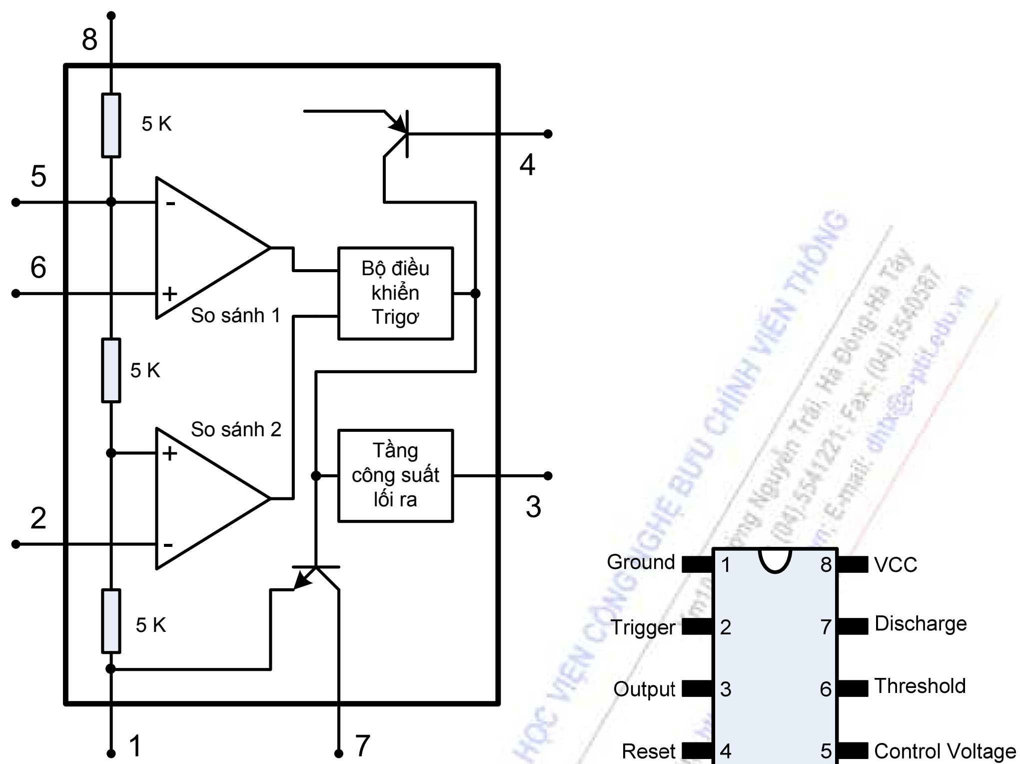
6.4. IC ĐỊNH THỜI

Bộ định thời 555 được sử dụng rất rộng rãi trong các bộ dao động đa hài, đa hài đợi, và các bộ so sánh v.v… Hình 6-17 là sơ đồ khối nguyên lý của IC định thời này, trong đó chức năng của các chân được chỉ ra trong bảng sau:

Chân	Chức năng	Chân	Chức năng
1	**Đất - GND**	**5**	**Điện áp điều khiển**
2	**Chân kích thích**	**6**	**Chân ngưỡng**
3	**Đầu ra**	**7**	**Đầu phóng điện**
4	**Xoá - Reset**	**8**	**Nguồn - Vcc**

Bảng chức năng của IC 555

TH	$\overline{TRIG}$	$\overline{R}$	OUT	DIS
X	X	L	L	Thông
$>\frac{2}{3}E_C$	$>\frac{1}{3}E_C$	H	L	Thông
$<\frac{2}{3}E_C$	$>\frac{1}{3}E_C$	H	Không đổi	Không đổi
X	$>\frac{1}{3}E_C$	H	H	Ngắt

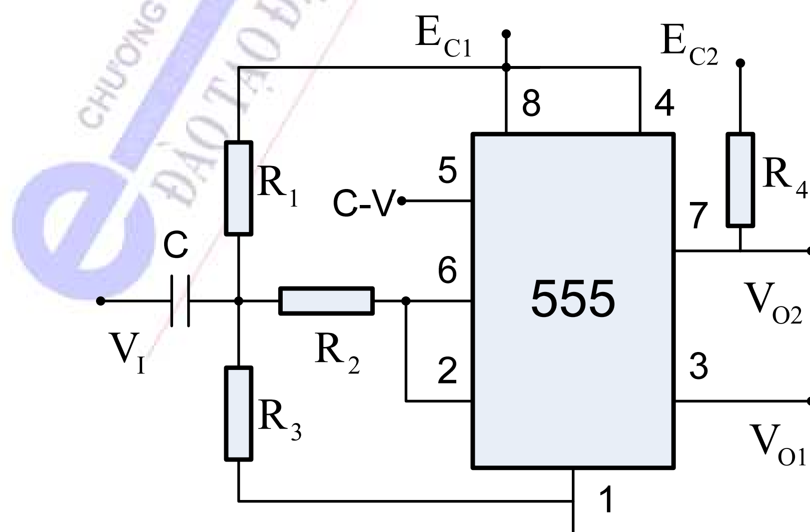


Hình 6-17. Sơ đồ khối nguyên lý của IC định thời 555

Một vài ứng dụng của IC định thời 555

1) Trigơ Schmitt

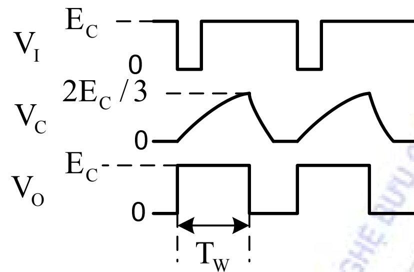
Hình 6-18 là sơ đồ nguyên lý của trigơ schmitt dùng IC 555. Với sơ đồ này ngưỡng trên $V_{T+} = \frac{2}{3} * E_{C1}$ và ngưỡng dưới $V_{T-} = \frac{1}{3} * E_{C1}$. Độ chênh lệch điện áp $\Delta V = V_{T+} - V_{T-} = \frac{1}{3} * E_{C1}$. Nếu đưa điện áp vào đầu vào C-V thì có thể điều chỉnh được V_{T+}, V_{T-} và ΔV.



Hình 6-18. Mạch trigơ Schmitt dùng IC 555

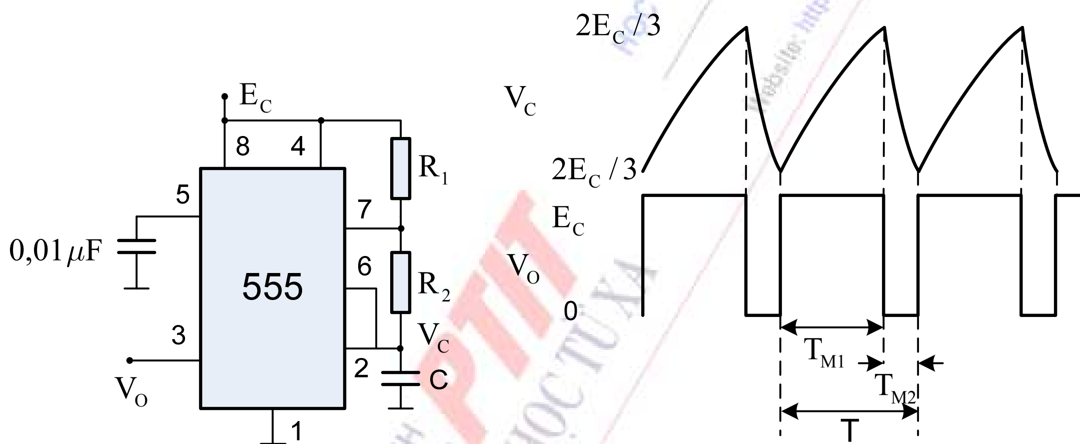
2) Mạch đa hài đợi

Hình 6-19 là sơ đồ nguyên lý và giản đồ thời gian của mạch đa hài đợi dùng IC 555, trong đó RC là mạch định thời. Độ kéo dài xung lối ra được xác định bằng công thức $T_W \approx RC\ln 3 \approx 1{,}1RC$. Mạch dao động đa hài đợi này yêu cầu độ rộng xung lối vào nhỏ hơn độ rộng xung lối ra, nếu nó lớn hơn thì yêu cầu dùng thêm mạch vi phân ở lối vào.



Hình 6-19. Mạch đa hài đợi dùng IC 555 và dạng sóng

3) Mạch đa hài



Hình 6-20. Mạch đa hài dùng IC 555 và dạng sóng

Hình 6-20 là sơ đồ mạch đa hài và dạng sóng, điện trở R1, R2 và tụ điện C đóng vai trò là mạch định thời. Chu kỳ đao động của tín hiệu lối ra được xác định thông qua thời gian phóng và nạp của tụ điện C như sau:

$$T_{M1} = (R_1 + R_2)C * \ln 2 = 0,7(R_1 + R_2)C$$

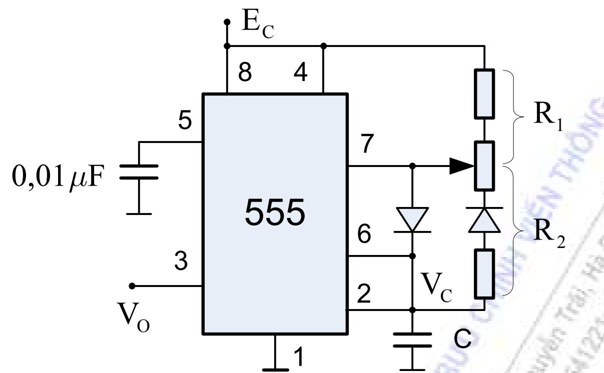
$$T_{M2} = R_2 C * \ln 2 = 0,7R_2 C$$

$$T = T_{M1} + T_{M2} = 0,7(R_1 + 2R_2)C$$

$$f = 1/T = \frac{1,43}{(R_1 + 2R_2)C}$$

Như ta thấy xung lối ra có độ lấp đầy phụ thuộc vào cả điện trở R1 và R2 và không thể tạo ra xung vuông với độ lấp đầy bằng 50% thông qua việc thay đổi giá trị R1 và R2. Để có được xung vuông với độ lấp đầy bằng 50%, người ta sử dụng mạch có thêm 2 diode khi đó trở phóng và

nạp điện cho Tụ có thể thay đổi độc lập và tạo ra xung mong muốn. Hình 6-21 là sơ đồ nguyên lý của mạch đa hài dùng IC 555 mà độ lấp đầy có thể thay đổi được.



Hình 6-21. Mạch đa hài điều chỉnh được độ lấp đầy xung dùng IC 555

TÓM TẮT

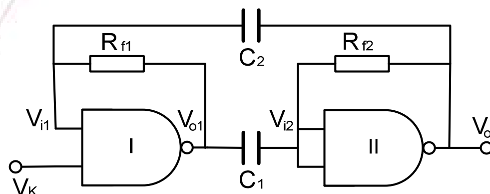
Trong chương này chúng ta đã tìm hiểu các mạch tạo xung. Mạch dao động xung tự kích không cần tín hiệu ngoài đưa vào; sau khi được cấp nguồn một chiều mạch tự động sinh ra xung vuôn. Thuộc loại dao động tự kích này có các mạch: bộ dao động đa hài cơ bản cổng NAND họ TTL, bộ dao động vòng, bộ dao động thạch anh, bộ dao động đa hài cơ bản CMOS.

Mạch tạo dạng xung không tự động phát xung nhưng có thể biến tín hiệu đàu vào hình dạng khác thành xung vuông theo yêu cầu của mạch số. Trong số mạch tạo dạng xung, chúng ta đã tìm hiểu: trigơ Schmit và đơn ổn.

Cách mạch phát xung và tạo dạng xung trên đây, ngoài dùng làm xung đồng hồ ra còn có ứng dụng vo cùng rộng rãi trong các hẹ thống xung - số. Bộ dao động đa hài thường dùng làm bộ tạo xung chuẩn thời gian và chuẩn tần số. Mạch đơn ổn thường dùng để định thời và làm trễ xung. Trigơ Schmit ngoài ứng dụng tạo dạng xung còn ứng dụng so sánh mức và giám sát mức…

CÂU HỎI ÔN TẬP

1. Trong mạch dao động đa hài cơ bản dùng cổng NAND họ TTL, hình 6-1, nếu giá trị trị điện trở $R_{f1} = 5*R_{f2} = 10\ k\Omega$, giá trị $C_1 = C_2 = 1\ \mu F$ thì mạch có hoạt động không? đạng tín hiệu tương đối lối ra sẽ như thế nào?



Hình 6-1. Bộ dao động đa hài cấu trúc bằng cổng NAND

a. Lối ra luôn ở mức logic thấp

b. Lối ra luôn ở mức logic cao

c. Tín hiệu lối ra là xung vuông với độ lấp đầy nhỏ hơn 50%

d. Tín hiệu lối ra là xung vuông có độ lấp đầy lớn hơn 50%

2. Với câu hỏi như câu 1 và giả thiết R_1= 3 kΩ, tính tần số dao động của mạch và vẽ dạng sóng lối ra.

a. f = 28 Hz và dạng sóng lối ra có dạng :

b. f=28 Hz và dạng sóng lối ra có dạng :

c. f=28 Hz và dạng sóng lối ra có dạng :

d. f=0 Hz và dạng sóng lối ra có dạng :

3. Đặc điểm nổi bật nhất của mạch dao động đa hài dùng thạch anh là gì?

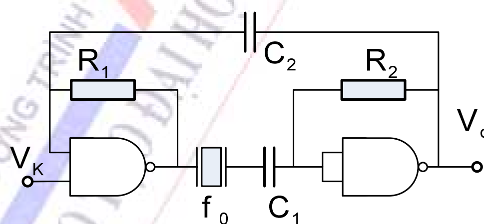
a. Biên độ tín hiệu lối ra ổn định

b. Tần số tín hiệu lối ra ổn định

c. Biên độ lối ra có thể điều chỉnh được

d. Tần số lối ra có thể điều chỉnh được

4. Trong mạch dao động đa hài dùng thạch anh như hình 6-6, nếu không có tụ C_1, lối ra của thạch anh được nối trực tiếp với dầu vào của cổng NAND thứ hai thì mạch:



Hình 6-6. Mạch dao động đa hài thạch anh

a. Không dao động lối ra luôn thấp

b. Không dao động lối ra luôn cao

c. Có xung lối ra nhưng tần số thay đổi

d. Tần số xung lối ra không thay đổi

5. Đặc điểm quan trọng nhất của trigơ Schmitt là gì?

a. Tần số hoạt động cao

b. Tính chống nhiễu cao vì nó hoạt động như bộ so sánh hai ngưỡng

c. Công suất tiêu thụ thấp

d. Là bộ so sánh một ngưỡng

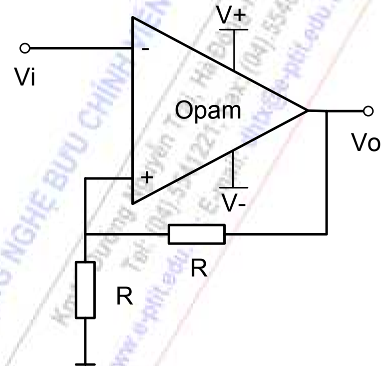
6. Mạch có sơ đồ nguyên lý như hình sau có chức năng như thế nào?

a. Bộ so sánh một ngưỡng

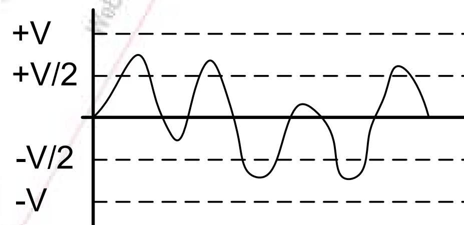
b. Trigơ Schmitt

c. Mạch dao động đa hài

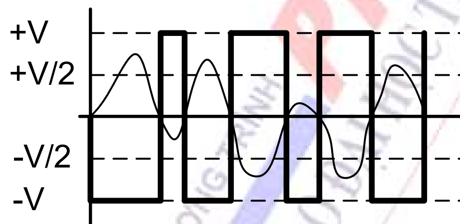
d. Mạch dao động đa hài đợi

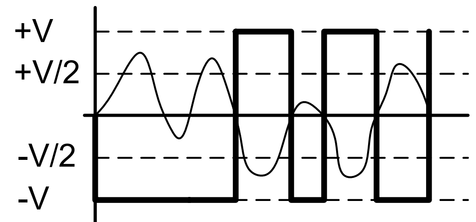


7. Với mạch điện như câu hỏi 6, nếu tín hiệu lối vào có dạng tín hiệu như hình sau, tín hiệu lối ra nằm ở hình nào.

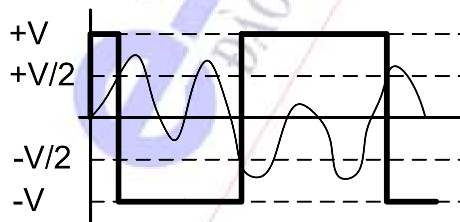


Hình a.

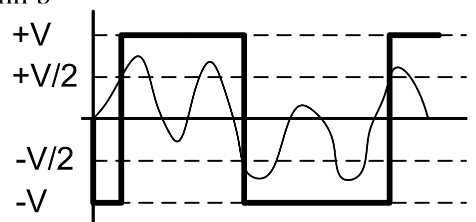




Hình b



Hình c



Hình d

a. Hình a.

b. Hình b.

c. Hình c.

d. Hình d.

8. Chức năng của mạch đa hài đợi là gì?

a. Là mạch phát xung vuông

b. Là mạch dao động đa hài có chân điều khiển

c. Là mạch dao động đa hài có một trạng thái ổn định và một trạng thái tạm ổn định

d. Là mạch phát xung điều hoà

9. Trong mạch đa hài đợi kiểu vi phân như hình 6-10, nếu xung điều khiển có độ rộng lớn hơn xung đa hài đợi lối ra thì :

a. Mạch vẫn hoạt động bình thường

b. Tín hiệu lối ra luôn bằng 0

c. Tín hiệu lối ra luôn bằng 1

d. Xung lối ra bằng xung lối vào

10. Trong mạch đa hài hình 6-20, nếu điện trở R_2 bị nối tắt thì:

a. Mạch vẫn phát xung và tần số lối ra chỉ phụ thuộc vào giá trị của R1 và C

b. Xung lối ra là xung vuông có độ lấp đầy là 50%

c. Mạch vẫn phát xung nhưng tần số rất cao

d. Không có tín hiệu lối ra

CHƯƠNG 7: BỘ NHỚ BÁN DẪN

GIỚI THIỆU

Bộ nhớ bán dẫn thay thế các loại bộ nhớ bằng vật liệu từ. Các tiến bộ mới của công nghệ bán dẫn trong thời gian gần đây đã cung cấp nhiều mạch nhớ loại MSI và LSI có độ tín cậy cao và giá thành hạ. Vào đầu thập kỷ 60 của thế kỷ 20, giá thành thương phẩm của một bit nhớ vào khoảng 2 USD. Đến nay (những năm đầu thế kỷ 21), giá thương phẩm của 128 Mbyte vào khoảng 20 USD. Như vậy giá thành thương phẩm của một bit nhớ sau khoảng 40 năm đã giảm đi khoảng 105.10^6 lần. Bộ nhớ bán dẫn điển hình có các tế bào nhớ sắp xếp theo hình chữ nhật, gắn trong khối hộp nhỏ bằng nhựa dạng DIP (Dual in line package). Tế bào nhớ cơ bản là một mạch trigơ, transistor hay mạch có khả năng tích trữ điện tích, tế bào nhớ này dùng để lưu trữ một bit tin.

Trong phần này giới thiệu một số bộ nhớ bán dẫn cơ bản.

NỘI DUNG

7.1. KHÁI NIỆM CHUNG

7.1.1. Khái niệm

Bộ nhớ là một thiết bị có khả năng lưu trữ thông tin (nhị phân). Muốn sử dụng bộ nhớ, trước tiên ta phải ghi dữ liệu và các thông tin cần thiết vào nó, sau đó lúc cần thiết phải lấy dữ liệu đã ghi trước đó để sử dụng. Thủ tục ghi vào và đọc ra phải được kiểm soát chặt chẽ, tránh nhầm lẫn nhờ định vị chính xác từng vị trí ô nhớ và nội dung của nó theo một mã địa chỉ duy nhất.

7.1.2. Những đặc trưng chính của bộ nhớ

7.1.2.1. Dung lượng của bộ nhớ.

Dung lượng bộ nhớ là số bit thông tin tối đa có thể lưu giữ trong nó. Dung lượng cũng có thể biểu thị bằng số từ nhớ n bit. ***Từ nhớ n bit*** là số bit (n) thông tin mà ta có thể đọc hoặc ghi đồng thời vào bộ nhớ. Ví dụ: Một bộ nhớ có dung lượng là 256 bit; nếu nó có cấu trúc để có thể truy cập cùng một lúcc 8 bit thông tin, thì ta cũng có thể biểu thị dung lượng bộ nhớ là 32 từ nhớ x 8 bit = 32 byte.

7.1.2.2. Cách truy cập thông tin.

Các bộ nhớ có thể có một trong hai cách truy cập thông tin.

Truy cập trực tiếp, hay còn gọi là truy cập ngẫu nhiên (random access). Ở cách này, không gian bộ nhớ được chia thành nhiều ô nhớ. Mỗi ô nhớ chứa được 1 từ nhớ n bit và có một địa chỉ xác định, mã hoá bằng số nhị phân k bit. Như vậy, người sử dụng có thể truy cập trực tiếp thông tin ở ô nhớ có địa chỉ nào đó trong bộ nhớ. Mỗi bộ nhớ có k bit địa chỉ sẽ có 2^k ô nhớ và có thể ghi được 2^k từ nhớ n bit.

Truy cập liên tiếp (serial access) hay còn gọi là kiểu truy cập tuần tự. Các đĩa từ, băng từ, trống từ, thanh ghi dịch…có kiểu truy cập này. Các bit thông tin được đưa vào và lấy ra một cách tuần tự.

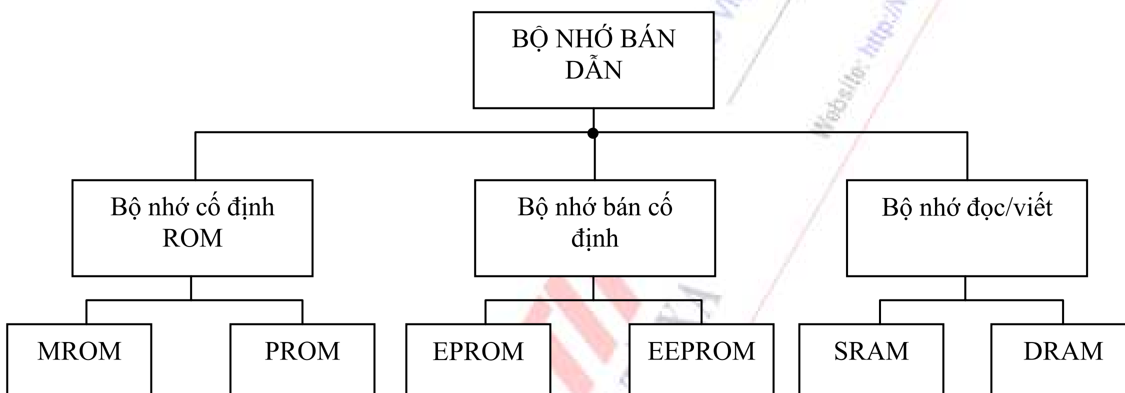
7.1.2.3. Tốc độ truy cập thông tin.

Đây là thông số rất quan trọng của bộ nhớ. Nó được đặc trưng bởi thời gian cần thiết để truy cập thông tin.

Thời gian truy cập thông tin ở các bộ nhớ truy cập kiểu trực tiếp gồm thời gian tìm địa chỉ của ô nhớ và thời gian đọc/viết thông tin trên đó. Thời gian truy cập thông tin phụ thuộc chủ yếu vào công nghệ chế tạo. Với công nghệ MOS thì thời gian truy cập khoảng 30 đến vài trăm ns.

Ở các bộ nhớ truy cập kiểu tuần tự, thời gian truy cập phụ thuộc vào vị trí của thông tin cần truy cập trong tập tin (file). Đối với các băng từ, đĩa từ thời gian truy cập của nó được định nghĩa là thời gian trung bình hoặc cực đại để truy cập một thông tin và nằm trong khoảng vài msec đến nhiều sec.

7.1.3. Phân loại



Dựa trên thời gian viết và cách viết, có thể chia thành bộ nhớ cố định, bộ nhớ bán cố định và bộ nhớ đọc/viết được. Bộ nhớ có nội dung được viết sẵn một lần khi chế tạo được gọi là bộ nhớ cố định và được ký hiệu là ROM (Read Only Memory). Sau khi đã được viết (bằng mặt nạ-mask) từ nhà máy thì ROM loại này không viết lại được nữa đó chính là MROM. PROM là một dạng khác, các bit có thể được viết bằng thiết bị ghi của người sử dụng trong một lần (Programmable ROM).

Bộ nhớ có thể đọc/ viết nhiều lần được gọi là RAM (Random Access Memory) gồm hai loại: bộ nhớ RAM tĩnh-SRAM (Static RAM) thường được xây dựng trên các mạch điện tử trigơ và RAM động-DRAM (Dynamic RAM) được xây dựng trên cơ sở nhớ các điện tích ở tụ điện; bộ nhớ này phải được hồi phục nội dung đều đặn, nếu không nội dung sẽ mất đi theo sự rò điện tích trên tụ. Giữa ROM và RAM có một lớp các bộ nhớ được gọi là EPROM (Erasable PROM), dữ liệu trong đó có thể xoá được bằng tia cực tím và ghi lại được, EEPROM (Electric EPROM) có thể xoá được bằng dòng điện. Các loại này còn được gọi là bộ nhớ bán cố định. Các bộ nhớ DRAM thường thoả mãn những yêu cầu khi cần bộ nhớ có dung lượng lớn; trong khi đó khi cần có tốc độ truy xuất lớn thì phải dùng các bộ nhớ SRAM có giá thành đắt hơn. Nhưng cả hai loại này đều có nhược điểm là thuộc loại "bay hơi" (volatile), thông tin sẽ bị mất đi khi nguồn nuôi bị

ngắt. Do vậy các chương trình dùng cho việc khởi động PC như BIOS thường phải nạp trên các bộ nhớ ROM.

7.1.4. Tổ chức của bộ nhớ

Bộ nhớ thường được tổ chức gồm nhiều vi mạch nhớ được ghép lại để có độ dài từ và tổng số từ cần thiết. Những chip nhớ được thiết kế sao cho có đầy đủ một số chức năng của bộ nhớ như:

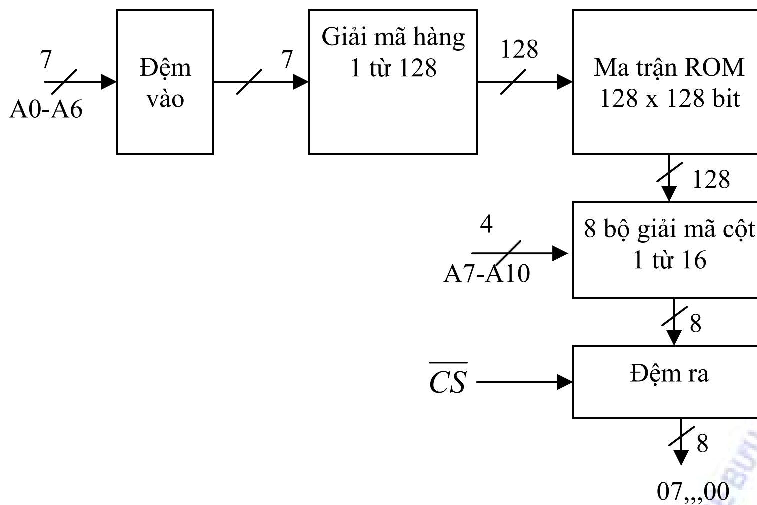
- Một ma trận nhớ gồm các ô nhớ, mỗi ô nhớ ứng với một bit nhớ.
- Mạch logic giải mã địa chỉ ô nhớ.
- Mạch logic cho phép đọc nội dung ô nhớ.
- Mạch logic cho phép viết nội dung ô nhớ.
- Các bộ đệm vào, bộ đệm ra và bộ mở rộng địa chỉ.

Cách tổ chức đơn giản nhất là tổ chức theo từ (word organized) với sự chọn tuyến tính. Một ma trận nhớ như vậy có độ dài của cột bằng số lượng từ W và độ dài của hàng bằng số lượng bit B trong một từ. Bộ chọn từ phải giải mã 1 từ W, nghĩa là giải mã để có một đầu ra duy nhất cho một từ trong bộ nhớ. Phương pháp này có thời gian truy nhập ngắn nhưng cần một bộ giải mã lớn khi tổng số từ lớn, do đó làm tăng giá thành sản phẩm.

Kích thước của phần giải mã địa chỉ sẽ giảm đi khi tổ chức ma trận nhớ và phần logic chọn từ cho phép giải mã hai bước. Ma trận nhớ sử dụng giải mã hai bước ứng với từ vật lý và từ logic. Từ vật lý bao gồm số lượng bit trong một hàng của ma trận. Từ logic bao gồm số lượng bit tương ứng với một từ logic được nhận biết và gửi ra cùng một lúc. Cần hai bộ giải mã: một bộ giải mã hàng để chọn một từ vật lý và một bộ giải mã cột gồm có một vài mạch hợp kênh chọn một từ logic từ một từ vật lý đã chọn. Một từ vật lý được chia thành S từ logic. Bộ giải mã hàng là bộ giải mã chọn 1 từ W mà B = W/S và bộ chọn cột chứa B bộ hợp kênh một đường từ S.

Ví dụ sơ đồ ROM dung lượng 2048 x 8 (2048 từ, mỗi từ chứa 8 bit) tổ chức giải mã hai bước như hình 7- 1.

Ma trận nhớ là 128 x 128, như vậy có 128 = 2^7 từ vật lý. Một từ vật lý được chọn bởi 7 đường địa chỉ từ A0 đến A6. Bộ giải mã hàng chọn 1 hàng từ 128 hàng. Một từ vật lý được chia thành 128/8 = 16 nhóm 8 bit. Nhóm thứ nhất chứa những bit có trọng số cao nhất của 16 từ logic. Nhóm thứ hai chứa các bit cao tiếp theo của 16 từ logic…Nhóm cuối cùng chứa những bit thấp nhất của 16 từ logic, do đó S = 16. Như vậy, những bộ giải mã cột gồm 8 bộ hợp kênh một đường từ 16 đường để cung cấp một từ locgic ra 8 bit. Những địa chỉ từ A7 đến A10 điều khiển các bộ giải mã cột. Trường hợp đặc biệt khi số phần tử trong một từ vật lý bằng sos bit trong một từ vật lý thì đó là bộ nhớ tổ chức theo bit có nghĩa là mỗi từ logic có độ dài 1 bit.



Hình 7-1. Một ví dụ về giải mã hai bước cho ma trận ROM 128 x 128

Các bộ đệm ra đảm bảo các mức logic mong muốn và cung cấp đủ dòng điện, ngoài ra nó còn có đầu ra collector hở hoặc 3 trạng thái cho phép nối chung đầu ra của một vài chip với nhau. Bộ đệm ra được điều khiển bởi một hay nhiều đầu vào như chọn mạch CS (Chip Select), cho phép mở CE (Chip Enable) hay cho phép mở đầu ba trạng thái OE (Output Enable).

7.2. DRAM

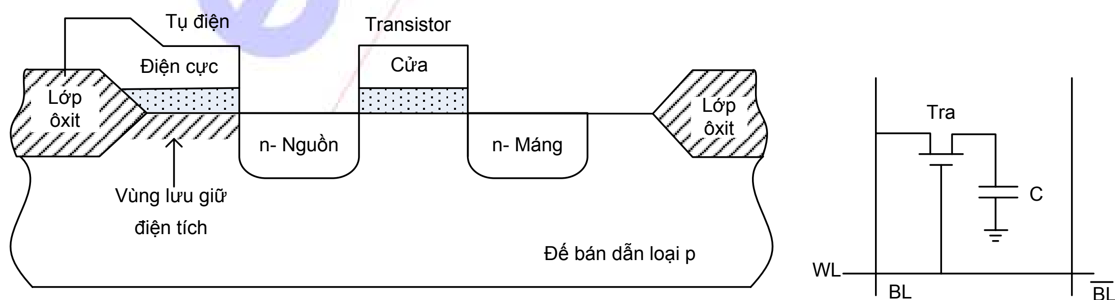
7.2.1. Cấu tạo của DRAM

Các ô nhớ được xắp xếp theo hàng và cột trong một ma trận nhớ. Địa chỉ ô nhớ được chia thành hai phần: địa chỉ hàng và cột. Hai địa chỉ này được đọc vào bộ đệm một cách lần lượt. Xử lý kiểu này được gọi là hợp kênh, lý do là để giảm kích thước bộ giải mã, tức là giảm kích thước và giá thành vi mạch. Quá trình dồn kênh địa chỉ này được điều khiển bởi các tín hiệu RAS (Row Access Strobe) và CAS (Column Access Strobe).

Nếu $\overline{RAS}$ ở mức tích cực thấp thì DRAM nhận được địa chỉ đặt vào nó và sử dụng như địa chỉ hàng.

Nếu $\overline{CAS}$ ở mức tích cực thấp thì DRAM nhận được địa chỉ đặt vào nó và sử dụng như địa chỉ cột.

Một ô nhớ của DRAM gồm có một transistor trường MOS có trở lối vào rất lớn và một tụ điện C là linh kiện lưu trữ một bit thông tin tương ứng với hai trạng thái có hoặc không có điện tích trên tụ.

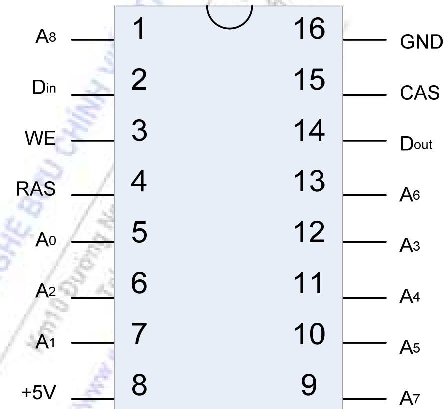


Hình 7-2. Cấu tạo một ô nhớ của DRAM

Transistor hoạt động như một công tắc, cho phép nạp hay phóng điện tích của tụ khi thực hiện phép đọc hay viết. Cực cửa (Gate) của transistor được nối với dây hàng (còn gọi là dây từ-WL-Word Line) và cực máng (Drain) được nối với dây cột (còn được gọi là dây bit BL hoặc $\overline{BL}$ - Bit Line), cực nguồn (Source) được nối với tụ điện. Điện áp nạp trên tụ tương đối nhỏ, vì thế cần sử dụng khuếch đại nhạy trong mạch nhớ. Do dòng rò của transistor nên ô nhớ cần được nạp lại trước khi điện áp trên tụ thấp hơn một ngưỡng nào đó. Quá trình này được thực hiện nhờ một chu kỳ "làm tươi" (refresh), khi đó điện áp trên tụ được xác định (ở trạng thái 0 hay 1) và mức điện áp logic này được viết lại vào ô nhớ.

Một số loại chip DRAM thường gặp là: TMS 4116: có dung lượng 16k x 1 bit; 41256 có dung lượng 256k x 1 bit. Thời gian truy cập thông tin khoảng 150 nsec, công suất tiêu thụ khoảng 280 mW khi làm việc (khi chờ = 28 mW)

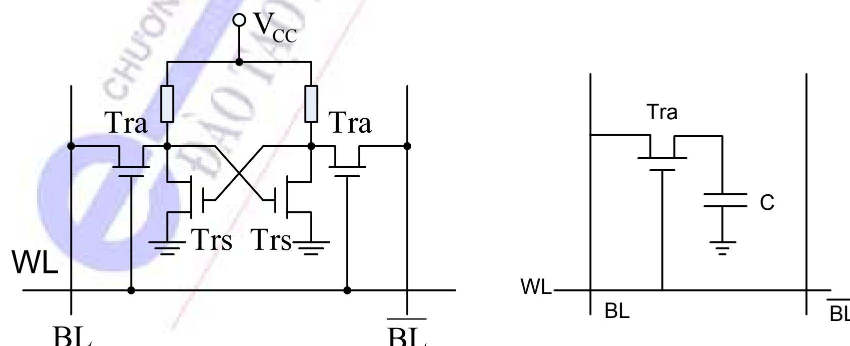
Hình 7-3 là vỏ của IC 41256 dung lượng 256k x 1 bit. Mạch cần 18 bit địa chỉ để mã hoá cho các địa chỉ hàng và cột; nhưng trên vỏ chỉ có 9 đương địa chỉ từ A_0 đến A_8. Hai chân RAS, CAS hoạt động ở mức cao, dùng để điều khiển 9 bit địa chỉ trên chip tới bộ giải mã địa chỉ hàng hay cột.



Hình 7-3. IC 41256

7.3. SRAM

Một ô nhớ của SRAM giữ thông tin bởi trạng thái của mạch trigơ. Thuật ngữ "tĩnh" chỉ ra rằng khi nguồn nuôi chưa bị cắt thì thông tin của ô nhớ vẫn được giữ nguyên. Khác với ô nhớ DRAM, ở đây ô nhớ trigơ cung cấp một tín hiệu số mạch hơn nhiều vì đã có các transistor trong các ô nhớ, chúng có khả năng khuếch đại tín hiệu và do đó có thể cấp trực tiếp cho các đường bit. Trong DRAM, sự khuếch đại tín hiệu trong các bộ khuếch đại cần nhiều thời gian và do đó thời gian truy nhập dài hơn. Khi định địa chỉ trong các trigơ ở SRAM, các transistor bổ sung cho các trigơ, các bộ giải mã địa chỉ…cũng được đòi hỏi như ở DRAM.



Hình 7-4. Cấu tạo một ô nhớ của SRAM và DRAM

Như trong DRAM, cực cửa của transistor được nối với đường từ và cực máng nối với cặp đường bit. Nếu số liệu được đọc từ ô nhớ, khi đó bộ giải mã hàng kích hoạt đường dây từ WL tương ứng. Hai transistor T dẫn và nối trigơ nhớ với cặp dây bit. Như vậy hai lối ra Q và $\overline{Q}$ được

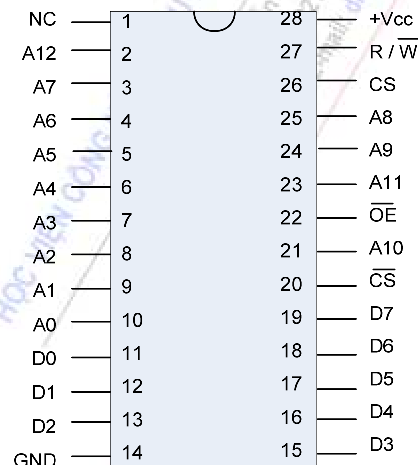
nối với các đường bit và các tín hiệu được truyền tới bộ khuếch đại ở cuối đường dây này. Vì điện thế chênh lệch lớn nên xử lý khuếch đại như vậy sẽ nhanh hơn trong DRAM (cỡ 10 ns hoặc ngắn hơn), do đó chip SRAM cần địa chỉ cột sớm hơn nếu thời gian truy nhập không được giảm. Như vậy SRAM không cần thực hiện phân kênh các địa chỉ hàng và cột. Sau khi số liệu ổn định, bộ giải mã cột chọn cột phù hợp và cho ra tín hiệu số liệu tới bộ đệm số liệu ra và tới mạch ra.

Viết số liệu được thực hiện theo cách ngược lại. Qua bộ đệm vào và bộ giải mã cột, số liệu viết được đặt vào bộ khuếch đại phù hợp. Cùng lúc đó bộ giải mã hàng kích hoạt đường dây từ và làm transistor T dẫn. Trigơ đưa số liệu được lưu trữ vào cặp dây bit. Tuy vậy, bộ khuếch đại nhạy hơn các transistor nên nó sẽ cấp cho các đường bit một tín hiệu phù hợp với số liệu viết. Do đó, trigơ sẽ chuyển trạng thái phù hợp với số liệu mới hoặc giữ giá trị đã được lưu trữ phụ thuộc vào việc số liệu viết trùng với số liệu đã lưu trữ hay không.

Một số IC DRAM thường gặp là 2148, 2114-2 của hãng Intel. Dung lượng 1k x 4 bit. Thời gian truy cập thông tin khoảng 200 ns, công suất tiêu thụ 525 mW.

IC TMS 4016 dung lượng 2k x 8 bit.

IC HM 6116, họ CMOS, dung lượng 2kbyte, thời gian truy cập là 120 nsec, công suất tiêu thụ khi làm việc là P = 180 mW (khi chờ ≈ μW). Hình 7-5 giới thiệu IC 6264, dung lượng 8 kbyte, và bảng điều kiện thao tác của nó.



Hình 7-5. Sơ đồ chân của SRAM 6264

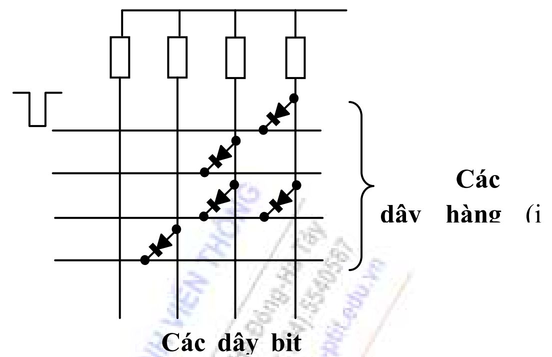
Phương thức hoạt động	$\overline{CS}$	CS	$\overline{WE}$	$\overline{OE}$
Không được chọn	H	X	X	X
Đọc	L	H	H	L
Đọc nhưng không xuất dữ liệu	L	H	H	H
Ghi	L	H	L	L

7.3. BỘ NHỚ CỐ ĐỊNH - ROM

Các chip RAM không thích hợp cho các chương trình khởi động do các thông tin trên đó bị mất khi tắt nguồn. Do vậy phải dùng đến ROM, trong đó các số liệu cần lưu trữ được viết một lần theo cách không bay hơi để nhằm giữ được mãi.

7.3.1. MROM

ROM lập trình theo kiểu mặt nạ được gọi là MROM. Nó được chế tạo trên một phiến silic theo một số bước xử lý như quang khắc và khếch tán để tạo ra những tiếp giáp bán dẫn có tính dẫn điện theo một chiều (như diode, transistor trường). Người thiết kế định rõ chương trình muốn ghi vào ROM, thông tin này được sử dụng để điều khiển quá trình làm mặt nạ. Hình 7-6 là một ví dụ đơn giản về sơ đồ MROM dùng diode.



Hình 7-6. MROM diode

Chỗ giao nhau giữa các dây từ (hàng) và các dây bit (cột) tạo nên một phần tử nhớ (ô nhớ). Một diode được đặt tại đó (hình vẽ) sẽ cho phép lưu trữ số liệu “0”. Ngược lại những vị trí không có diode thì sẽ cho phép lưu trữ số liệu “1”. Khi đọc một từ số liệu thứ i của ROM, bộ giải mã sẽ đặt dây từ đó xuống mức logic thấp, các dây còn lại ở mức cao. Do vậy chỉ những diode nối với dây này được phân cực thuận, do đó nó sẽ dẫn làm cho điện thế lối ra trên các dây bit tương ứng ở mức logic thấp, các dây bit còn lại sẽ giữ ở mức cao.

Cả hai công nghệ MOS và lưỡng cực được dùng để chế tạo MROM. Thời gian truy nhập của bộ nhớ lưỡng cực khoảng từ 50 – 90 ns, bộ nhớ MOS lâu hơn khoảng 10 lần. Do đó ROM lưỡng cực nhanh hơn và có khả năng kích hoạt tốt hơn trong khi mạch nhớ MOS cùng dung lượng có kích thước nhỏ hơn và tiêu thụ năng lượng ít hơn.

7.3.2. PROM

PROM cũng gồm có các diode như ở MROM nhưng chúng có mặt đầy đủ tạo các vị trí giao nhau giữa dây từ và dây bit. Mỗi diode được nối với một cầu chì. Bình thường khi chưa lập trình, các cầu chì còn nguyên vẹn, nội dung của PROM sẽ toàn là 0. Khi định vị đến một bit bằng cách đặt một xung điện ở lối ra tương ứng, cầu chì sẽ bị đứt và bit này sẽ bằng 1. Bằng cách đó ta có thể lập trình toàn bộ các bit trong PROM. Như vậy, việc lập trình đó có thể được thực hiện bởi người sử dụng chỉ một lần duy nhất, không thể sửa đổi được.

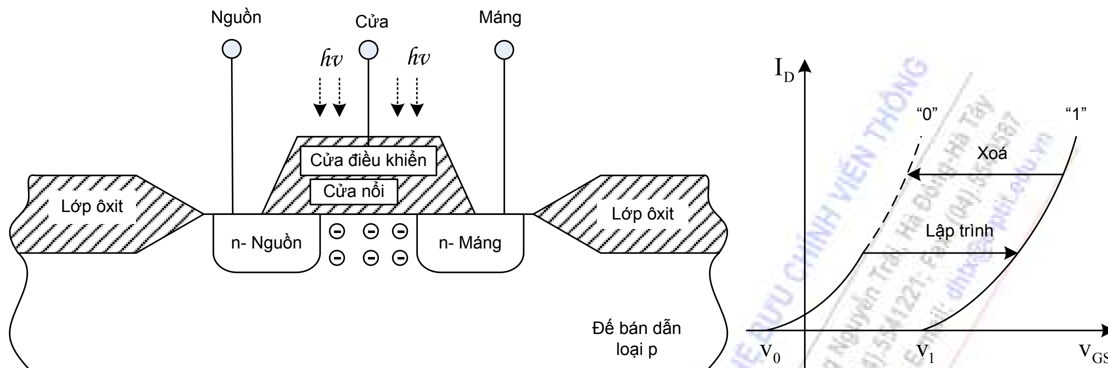
7.4. BỘ NHỚ BÁN CỐ ĐỊNH

7.4.1. EPROM (Erasable PROM)

Số liệu vào có thể được viết vào bằng xung điện nhưng được lưu giữ theo kiểu không bay hơi. Đó là loại ROM có thể lập trình được và xóa được. Hình 7- 7 chỉ ra cấu trúc của một transistor dùng để làm một ô nhớ gọi là FAMOST (Floating gate avalanche injection MOS transistor).

Trong ô nhớ dùng transistor này, cực cửa được nối với đường từ, cực máng được nối với đường bit và cực nguồn được nối với nguồn chuẩn được coi là nguồn cho mức logic 1. Khác với transistor MOS bình thường, transistor loại này còn có thêm một cửa gọi là *cửa nổi* (floating gate); đó là một vùng vật liệu được thêm vào vào giữa lớp cách điện cao như ở hình 7-7. Nếu cửa nổi không có điện tích thì nó không ảnh hưởng gì đến cực cửa điều khiển và transistor hoạt động như bình thường. Tức là khi dây từ được kích hoạt (cực cửa có điện thế dương) thì transtor dẫn, cực máng và nguồn được nối với nhau qua kênh dẫn và dây bit có mức logic 1. Nếu cửa nổi có các điện tử trong đó với điện tích âm thì chúng sẽ ngăn trường điều khiển của cửa cửa và dù dây

từ được kích hoạt thì cũng không thể phát ra trường đủu mạnh với cực cửa điều khiển để làm thông transistor. Lúc này đường bit không được nối với nguồn chuẩn và ô nhớ coi như được giữ giá trị 0.



Hình 7-7. Cấu trúc của một EPROM

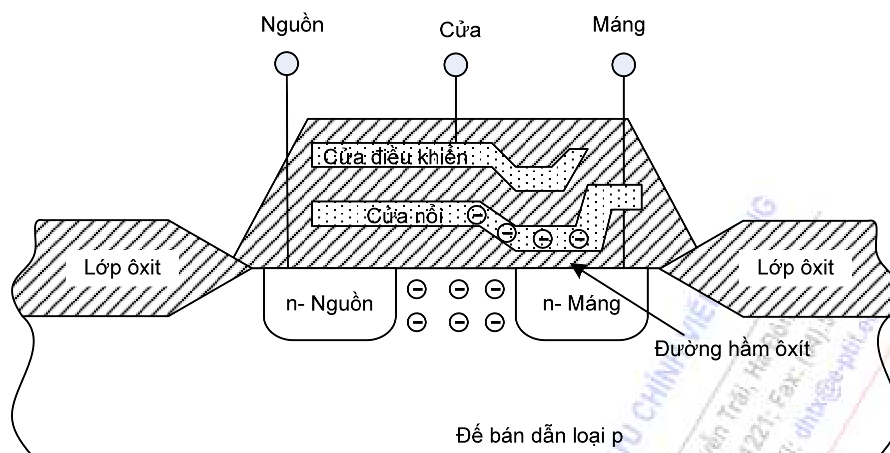
Việc nạp các điện tử vào vùng cửa nổi, tức là tạo ra các ô nhớ mang giá trị 0 được thực hiện bởi xung điện có độ dài cỡ 50 ms và độ lớn + 20 V đặt giữa cực cửa va cực máng. Lúc đó những điện tích mang năng lượng lớn sẽ đi qua lớp cách điện giữa đế và cửa nổi. Chúng tích tụ trong vùng cửa nổi và được giữ ở đây sau khi xung lập trình tắt. Đó là do cửa nổi được cách điện cao với xung quanh và các điện tử không còn đủ năng lượng sau khi lạnh đi, để có thể vượt ra ngoài lớp cách điện đó nữa. Chúng sẽ được giữ ở đây trong một thời gian rất dài (ít nhất là 10 năm).

Để xoá các thông tin, tức là làm mất các điện tích điện tử trong vùng cửa nổi, phải chiếu ánh sáng tử ngoại UV vào chíp nhớ. Lúc này, những điện tử hấp thụ đượ năng lượng và sẽ nhảy lên các mức năng lượng cao và rời khỏi cửa nổi giống như cách mà chúng đã thâm nhập vào. Trong chip EPROM có một cửa sổ làm bằng thuỷ tinh thạch anh chỉ để cho ánh sáng tử ngoại đi qua khi cần xoá số liệu trong bộ nhớ.

7.4.2. EEPROM (Electrically Erasable PROM)

Cửa sổ thạch anh có giá thành khá đắt và không tiện lợi nên những năm gần đây xuất hiện các chip PROM có thể xoá số liệu bằng phương pháp điện. Cấu trúc của ô nhớ giống như hình 7-8.

Việc nạp các điện tử cho cửa nổi được thực hiện như cách ở EPROM. Bằng một xung điện tương đối dài, các điện tích mang năng lượng cao được phát ra trong đế sẽ thấm qua lớp cửa ôxit và tích tụ trong cửa nổi. Để xoá EEPROM, một lớp kênh màng mỏng ôxit giữa vùng cửa nổi trải xuống dưới đế và cực máng giữ vai trò quan trọng. Các lớp cách điện không thể là lý tưởng được, các điện tích có thể thấm qua lớp phân cách với một xác suất thấp. Xác suất này tăng lên khi bề dày của lớp giảm đi và điện thế giữa hai điện cực ở hai mặt lớp cách điện tăng lên. Muốn phóng các điện tích trong vùng cửa nổi một điện thế (-20 V) được đặt vào cực cửa điều khiển và cực máng. Lúc này các điện tử âm trong cửa nổi được chảy về cực máng qua kênh màng mỏng ôxit và số liệu lưu giữ được xoá đi. Điều lưu ý là phải làm sao cho dòng điện tích này chảy không quá lâu vì nếu không vùng cửa nổi này lại trở nên tích điện dương làm cho hoạt động của transistor không được trạng thái bình thường (mức nhớ 1).



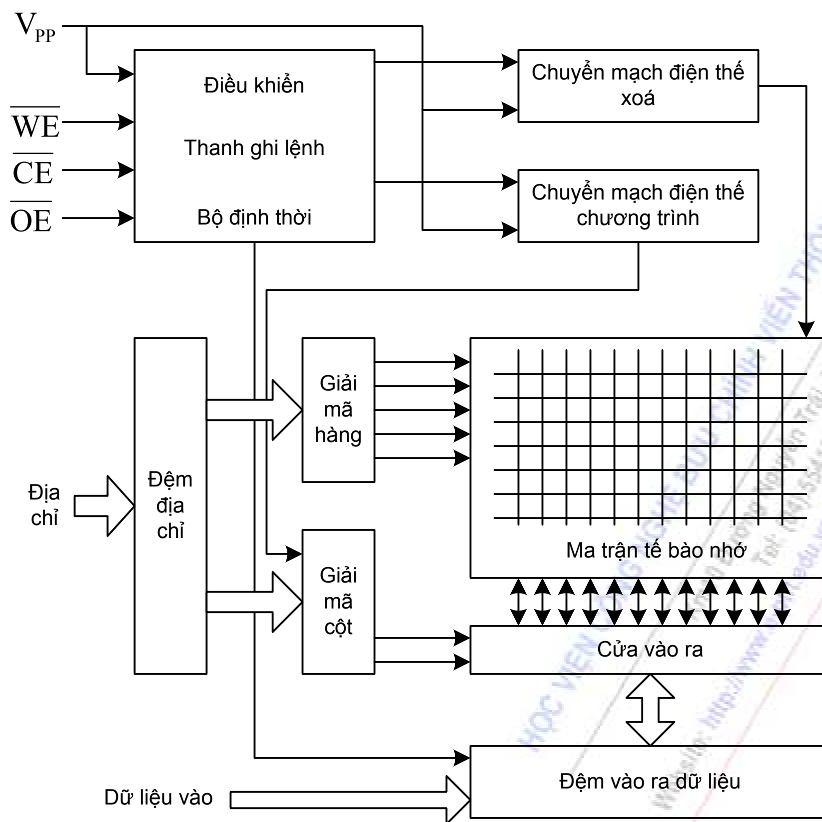
Hình 7-8. Cấu trúc của EEPROM

Các chip ROM hiện nay có thời gian truy nhập từ 120 ns đến 150 ns dài hơn nhiều thời gian đó trong các chip nhớ RAM.

7.4.3. Đĩa cứng silicon- Bộ nhớ FLASH

Trong những năm gần đây, một loại bộ nhớ không bay hơi mới đã xuất hiện trên thị trường, thường được sử dụng thay thế cho các ổ đĩa mềm và cứng trong những máy tính. Đó là bộ nhớ flash. Cấu trúc của chúng cơ bản như EEPROM, chỉ có lớp kênh ôxit ở các ô nhớ mỏng hơn. Do vậy chỉ cần điện thế cỡ 12 V là có thể cho phép thực hiện 10 000 chu trình xoá và lập trình. Bộ nhớ flash có thể hoạt động gần mềm dẻo như DRAM và SRAM nhưng lại không bị mất số liệu khi bị cắt điện. Hình 7- 9 chỉ ra sơ đồ khối của nó.

Phần chính là mạng nhớ bao gồm các ô nhớ FAMOST như được mô tả ở mục trên. Giống như SRAM, bộ nhớ flash không dồn phân kênh địa chỉ. Các bộ giải mã hàng và cột chọn một đường từ và một hoặc nhiều cặp đường bit. Số liệu đọc được đưa ra ngoài bộ đệm số liệu I/O hoặc được viết vào ô nhớ đã được định địa chỉ bởi bộ đệm này qua cổng I/O. Xử lý đọc được thực hiện với điện thế MOS thông thường là 5V. Để lập trình một ô nhớ, đơn vị điều khiển flash đặt một xung điện thế ngắn cỡ 10 µs và 12 V gây nên một sự chọc thủng thác lũ vào transistor nhớ để nạp vào cửa nổi. Một chip nhớ flash 1 Mb có thể được lập trình trong khoảng 2 sec, nhưng khác với EEPROM việc xoá được thực hiện từng chip một. Thời gian xoá cho toàn bộ bộ nhớ flash khoảng 1 sec. Xử lý đọc, lập trình và xoá được điều khiển bởi các lệnh có độ dài 2 byte được bộ xử lý viết vào các thanh ghi lệnh của mạch điều khiển flash.



Hình 7-9. Sơ đồ bộ nhớ FLASH

Mục đích sử dụng chính của bộ nhớ flash là để thay thế cho các ổ đĩa mềm và ổ đĩa cứng dung lượng nhỏ. Do nó là mạch tích hợp nên có ưu điểm là kích thước nhỏ và tiêu thụ năng lượng thấp, không bị ảnh hưởng của va đập. Các đĩa cứng chất rắn dựa trên cơ sở các bộ nhớ flash có lợi thế về công suất tiêu thụ cũng như giá thành có dung lượng tới vài Mbyte. Các card nhớ loại này có ưu điểm là không gặp phải vấn đề mất thông tin như trường hợp RAM CMOS khi pin Ni-Cd bị hỏng. Thời gian lưu trữ thông tin trong bộ nhớ flash ít nhất là 10 năm, thông thường là 100 năm, với khoảng thời gian này thì các đĩa mềm và cứng đã bị hỏng rồi.

Nhược điểm của bộ nhớ flash là chỉ có thể xoá theo kiểu lần lượt từng chip hoặc lần lượt từng trang.

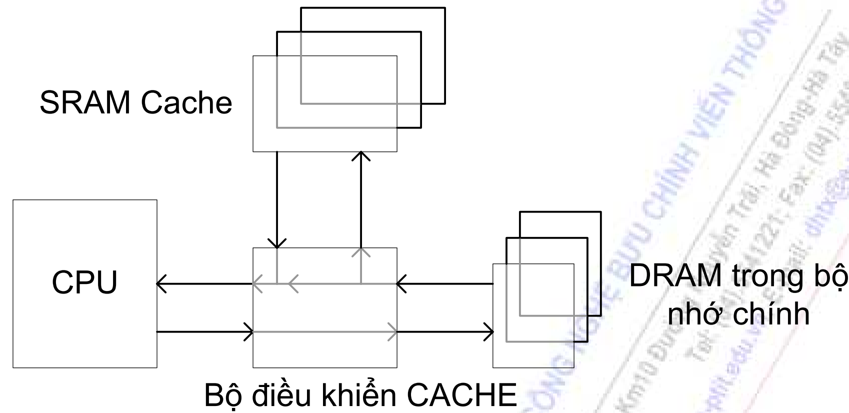
7.4.3. Bộ nhớ CACHE

Với các máy tính có tốc độ nhanh (trên 33MHz), cần phải xen các trạng thái đợi khi truy xuất dữ liệu tới các DRAM rẻ tiền nhưng có thời gian thâm nhập chậm (60-120ns). Điều này làm giảm hiệu suất của máy. Có thể giải quyết bằng cách dùng các SRAM có thời gian thâm nhập ngắn hơn (20-25 ns, thậm chí 12 ns) nhưng giá thành lại rất đắt. Bộ nhớ Cache kết hợp được các lợi điểm nhanh của SRAM và rẻ của DRAM. Giữa CPU và bộ nhớ chính bằng DRAM, người ta xen vào một bộ nhớ SRAM nhanh có dung lượng nhỏ bằng 1/10 hoặc 1/100 lần bộ nhớ chính gọi là cache; dưới sự điều khiển của mạch điều khiển cache, bộ nhớ này sẽ lưu trữ tạm thời các số liệu thường được gọi và cung cấp nó cho CPU trong thời gian ngắn.

Cache chứa các thông tin mới vừa được CPU sử dụng gần đây nhất. Khi CPU đọc số liệu nó sẽ đưa ra một địa chỉ tới bộ điều khiển cache. Sau đó một trong hai quá trình sau sẽ xảy ra:

- Cache hit: nếu địa chỉ đó đã có sẵn trong RAM cache.
- Cache miss: ngược lại, nếu địa chỉ đó không có sẵn trong RAM cache.

Như vậy, cache hit tỷ lệ với truy xuất thông tin có sẵn trong bộ nhớ cache SRAM, còn cache miss lại tỷ lệ với truy xuất thông tin có trong bộ nhớ chính là các DRAM.



Hình 7-10. Nguyên lý của Cache

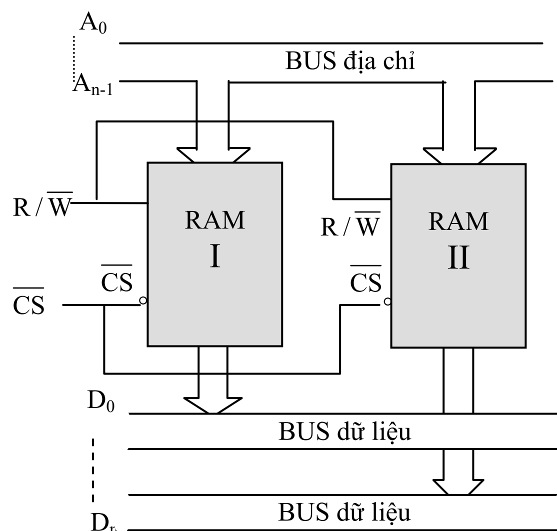
7.5. MỞ RỘNG DUNG LƯỢNG BỘ NHỚ

Các vi mạch nhớ bán dẫn chỉ có dung lượng xác định. Muốn có bộ nhớ có dung lượng lớn hơn, ta tìm cách ghép nhiều vi mạch nhớ nhằm một trong ba mục đích sau:

- Tăng độ dài nhớ, nhưng không làm tăng số lượng từ nhớ.
- Tăng số lượng từ nhớ nhưng không làm tăng độ dài từ nhớ.
- Tăng cả số lượng và độ dài từ nhớ.

7.5.1 Mở rộng độ dài từ

Trên một chíp nhớ, có thể có được 1 đến một số hữu hạn lối ra, thường là 4 hoặc 8 bit. Muốn có độ dài từ lớn hơn, chẳng hạn từ 4 lên 8 hoặc 16 bit, ta tiến hành ghép nhiều chíp nhớ như chỉ ở hình 7-10 đối với RAM. Đối với ROM cách làm cũng tương tự, chỉ khác trong trường hợp này, có thể không có lối vào R/ $\overline{W}$.



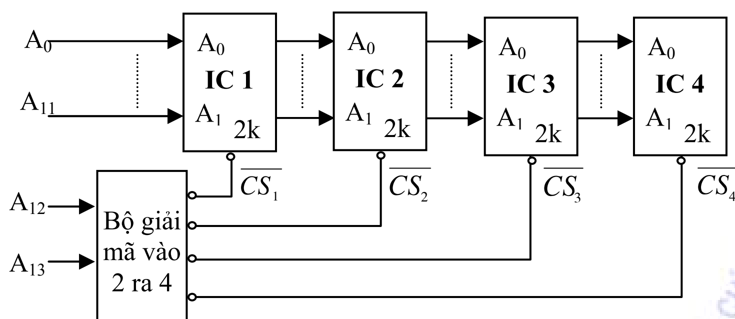
Hình 7-10. Sơ đồ mở rộng độ dài từ.

7.5.2 Mở rộng dung lượng

Muốn mở rộng dung lượng, ta cũng ghép nhiều chíp lại với nhau. Như đã biết, dung lượng có liên quan đến số lối vào địa chỉ (C = 2N x độ dài từ, với N là số lối vào địa chỉ). Cứ tăng 1 chíp thì cần có thêm một lối vào địa chỉ.

Khác với trường hợp mở rộng độ dài từ, khi mở rộng dung lượng các lối vào/ra dữ liệu D và

$R/\overline{W}$ được nối song song. Một phần dung lượng được trữ vào mỗi chíp. Sự phân chia này dựa trên cơ sở tổ hợp địa chỉ vào và lối vào điều khiển. Hình 7-11 là một sơ đồ ví dụ.



Hình 7-11. Phương pháp mở rộng dung lượng.

Để thực hiện phép mở rộng ta phải sử dụng một số lối vào địa chỉ dành riêng cho bộ giải mã (thường là các địa chỉ có trọng số cao). Ở sơ đồ trên ta chọn 2 địa chỉ A_{12} và A_{13} để giải mã. Do đó ta có thể nhận được 4 giá trị ra tương ứng. Các giá trị này tác động lên các lối vào CS để mở tuần tự các IC nhớ. Các IC nhớ này có thể làm ROM hoặc RAM hoặc cả hai là tùy chọn. Tuần tự mở các IC theo A_{12}, A_{13} như chỉ ra ở bảng hoạt động sau.

A_{13}	A_{12}	$\overline{CS}$	IC mở	Khoảng địa chỉ
0	0	$\overline{CS_1}$	IC I	0000_{16} - $0FFF_{16}$
0	1	$\overline{CS_2}$	IC II	1000_{16} - $1FFF_{16}$
1	0	$\overline{CS_3}$	IC III	2000_{16} - $2FFF_{16}$
1	1	$\overline{CS_4}$	IC IV	3000_{16} - $3FFF_{16}$

Kỹ thuật này thường được ứng dụng trong các hệ thống vi xử lý, phổ biến nhất là các máy vi tính. Phương pháp này không chỉ cho phép mở rộng dung lượng, mà còn tạo ra sự phân vùng nhớ. Chỉ cần ba địa chỉ giải mã đã có thể tạo ra được 8 vùng nhớ với dung lượng tùy thuộc các chíp thành phần.

TÓM TẮT

Trong chương này chúng ta trình bày nguyên lý cấu tạo, các tính năng cơ bản của các loại bộ nhớ bán dẫn: ROM, PROM, EPROM, EEPROM, SRAM, DRAM, FLASH, CACHE.

Các chip RAM không thích hợp cho các chương trình khởi động do các thông tin trên đó bị mất khi tắt nguồn. Do vậy phải dùng đến ROM, trong đó các số liệu cần lưu trữ được viết một lần theo cách không bay hơi để nhằm giữ được mãi.

Trong những năm gần đây, một loại bộ nhớ không bay hơi mới đã xuất hiện trên thị trường, thường được sử dụng thay thế cho các ổ đĩa mềm và cứng trong những máy tính. Đó là bộ nhớ flash. Cấu trúc của chúng cơ bản như EEPROM, chỉ có lớp kênh ôxit ở các ô nhớ mỏng hơn.

Với các máy tính có tốc độ nhanh (trên 33MHz), cần phải xen các trạng thái đợi khi truy xuất dữ liệu tới các DRAM rẻ tiền nhưng có thời gian thâm nhập chậm (60-120ns). Điều này làm giảm hiệu suất của máy. Có thể giải quyết bằng cách dùng các SRAM có thời gian thâm nhập

ngắn hơn (20-25 ns, thậm chí 12 ns) nhưng giá thành lại rất đắt. Bộ nhớ Cache kết hợp được các lợi điểm nhanh của SRAM và rẻ của DRAM.

Trong chương này còn giới thiệu cách mở rộng dung lượng và độ dài từ của bộ nhớ bán dẫn.

CÂU HỎI ÔN TẬP

1. Bộ nhớ ROM là bộ nhớ:
   - a. Chỉ có thể đọc.
   - b. Chỉ có thể viết.
   - c. Có thể vừa đọc vừa viết.
   - d. Không có phương án nào đúng.
2. Bộ nhớ RAM là bộ nhớ:
   - a. Chỉ có thể đọc.
   - b. Chỉ có thể viết.
   - c. Có thể vừa đọc vừa viết.
   - d. Không có phương án nào đúng.
3. Linh kiện lưu giữ bit thông tin của DRAM là:
   - a. Transistor.
   - b. Trigơ.
   - c. Tụ điện.
   - d. Diode.
4. Linh kiện lưu giữ bit thông tin của SRAM là:
   - a. Transistor.
   - b. Trigơ.
   - c. Tụ điện.
   - d. Diode.
5. MROM được chế tạo bởi công nghệ :
   - a. Lưỡng cực.
   - b. MOS.
   - c. Lưỡng cực và MOS.
   - d. Không có phương án nào đúng.
6. PROM là loại ROM có thể:
   - a. Chỉ lập trình được một lần.
   - b. Lập trình được nhiều lần.

c. Lập trình được và xoá được.

d. Không có phương án nào đúng.

7. Linh kiện lưu giữ bit thông tin của EPROM là:

a. Transistor lưỡng cực.

b. Transistor trường.

c. Tụ điện.

d. Diode.

8. Trong EPROM, việc nạp các điện tích vào vùng cửa nổi có nghĩa là:

a. Tạo các ô nhớ mang giá trị 0.

b. Tạo các ô nhớ mang giá trị 1.

c. Tạo các ô nhớ mang giá trị 0 và 1.

d. Không có phương án nào đúng.

9. EEPROM là loại ROM có thể:

a. Chỉ lập trình được một lần.

b. Lập trình được và xoá được một lần.

c. Lập trình được và xoá được nhiều lần.

d. Không có phương án nào đúng.

10. Muốn xoá dữ liệu trong EEPROM thì cần:

a. Chiếu tia tử ngoại vào.

b. Cần đặt vào cực cửa điều khiển và cực máng một điện thế có giá trị 20V.

c. Cần đặt vào cực cửa điều khiển và cực máng một điện thế có giá trị - 20V.

d. Cả phương án trên đều đúng.

11. Bộ nhớ FLASH là loại bộ nhớ:

a. Mất dữ liệu khi mất nguồn nuôi.

b. Không mất dữ liệu khi mất nguồn nuôi.

c. Bị mất dần dữ liệu ngay cả khi có nguồn nuôi.

d. Không có phương án nào đúng.

12. Bộ nhớ FLASH là loại bộ nhớ có thể thay thế cho:

a. Ổ đĩa mềm.

b. Ổ đĩa cứng.

c. Ổ mềm và ổ cứng có dung lượng nhỏ.

d. Không có phương án nào đúng.

CHƯƠNG 8: LOGIC LẬP TRÌNH (PLD)

GIỚI THIỆU

Các mạch kỹ thuật số tổ hợp và tuần tự đã được đề cập ở các chương trước. Các IC số rất đa dạng từ thực hiện các phép tính kỹ thuật số căn bản đến các chức năng phức tạp khác như: bộ hợp kênh, phân kênh, bộ cộng, so sánh, bộ mã hoá, giải mã, bộ đếm… Chúng là các IC số có chức năng cố định, tức là mỗi IC thực hiện một chứ năng chuyên biệt. Những linh kiện này được sản xuất một số lượng lớn để đáp ứng nhu cầu ứng dụng phong phú.

Để thiết kế một mạch, nhà thiết kế có thể chọn từ các IC có sẵn phù hợp nhất cho mạch điện. Phần thiết kế này có thể được chỉnh sửa để đáp ứng các yêu cầu chuyên biệt của những linh kiện này.

Ưu điểm của phương pháp này là:

1. Chi phí phát triển thấp.
2. Vận hành nhanh xung quanh bản thiết kế.
3. Tương đối dễ thử nghiệm các mạch

Nhược điểm:

1. Các yêu cầu về kích thước trong bảng mạch lớn.
2. Yêu cầu về điện lớn.
3. Thiếu tính bảo mật. (Các bảng mạch có thể bị sao chép).
4. Các yêu cầu về chi phí bổ sung, khoảng trống, điện…cần thiết để chỉnh sửa bản thiết kế hoặc trình bày các tính năng khác.

Để khắc phục những nhược điểm của thiết kế bằng cách sử dụng các IC chức năng cố định, các mạch tích hợp chuyên biệt ứng dụng (ASIC-Aplication Specific IC) đã được phát triển. Các ASIC đã được thiết kế để đáp ứng các yêu cầu chuyên biệt của một mạch và được giới thiệu bởi một nhà sản xuất IC. Các thiết kế này quá phức tạp không thể thực hiện bằng cách sử dụng các IC chức năng cố định được.

Ưu điểm của phương pháp này là:

1. Giảm thiểu được kích thước thông qua việc sử dụng mức tích hợp cao.
2. Giảm thiểu được yêu cầu về điện.
3. Nếu được sản xuất theo một quy mô lớn thì chi phí giảm đáng kể.
4. Việc thiết kế được thực thi dưới dạng này thì hoàn toàn không thể sao chép được.

Nhược điểm:

1. Chi phí phát triển ban đầu có thể cực kỳ lớn.

2. Các phương pháp thử nghiệm phải được phát triển và điều này làm gia tăng chi phí và công sức.

Có một phương pháp khác có các ưu điểm của hai phương pháp trên là sử dụng các thiết bị logic có thể lập trình được (PLD). Một thiết bị logic có thể lập trình là một IC mà người dùng có thể cấu hình để chúng có khả năng thực thi các chức năng logic như mong muốn. Đây là một chip LSI có chứa một cấu trúc "bình thường" và cho phép nhà thiết kế tạo tuỳ biến cho nó để dùng cho bất kỳ ứng dụng đặc biệt nào, tức là nó có thể được người dùng lập trình để thực hiện một chức năng cần thiết cho ứng dụng của họ.

Các PLD có các ưu điểm sau:

1. Chu kỳ thiết kế ngắn.
2. Chi phí phát triển thấp.
3. Giảm thiểu được yêu cầu khoảng trống trên bảng mạch.
4. Giảm thiểu được yêu cầu về điện.
5. Bảo đảm tính bảo mật của thiết kế.
6. Mạch được kết chặt lại.
7. Tốc độ đảo mạch nhanh hơn.
8. Mật độ tích hợp cao.
9. Chi phí sản xuất số lượng lớn thấp.

PLD cũng cho phép nhà thiết kế có nhiều phương tiện linh động hơn để thí nghiệm với các bản thiết kế bởi vì chúng có thể được lập trình lại trong vài giây.

Với nhiều ưu điểm như vậy nên hiện nay có một số lượng lớn các PLD được các nhà sản xuất IC tạo ra với nhiều tính năng đa dạng và nhiều tuỳ chọn có sẵn để nhà thiết kế mạch có thể sử dụng một cách phổ biến. Cấu trúc và các tính năng đa dạng khác của các PLD như ROM, các mảng logic lập trình (PLA). Logic mảng có thể lập trình (PAL), thiết bị logic có thể lập trình đơn giản (SPLD), và các mảng cổng có thể lập trình trường (FPGA) sẽ được đề cập ở đây. Công dụng của những thiết bị này yêu cầu phải có thay đổi thiết kế truyền thống, mặc dầu các khái niệm cơ bản vẫn được giữ lại không đổi.

NỘI DUNG

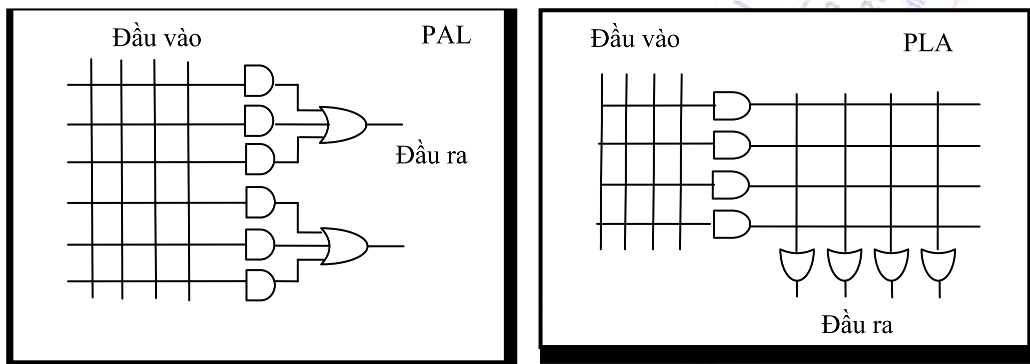
8.1. GIỚI THIỆU CHUNG VỀ LOGIC KHẢ TRÌNH (PLD)

Vi mạch lập trình, viết tắt là PLD (Programmable Logic Device), là loại cấu kiện điện tử có nhiều ưu điểm và hiện nay đang được phát triển rất mạnh. Về nguyên lý, chúng có cấu tạo rất giống với PROM. Việc lập trình cho PLD có thể được thực hiện bằng các công nghệ khác nhau, dựa trên cơ sở bẻ cầu chì hoặc chuyển mạch. Tuy nhiên, ứng dụng của PLD lại rất khác với PROM. Một PLD, được tạo thành bằng một số cổng AND, OR, XOR hoặc cả các trigơ, có thể thực hiện nhiều hàm Boole khác nhau.

8.2 SPLD

SPLD - cấu kiện logic khả trình đơn giản. Đây là loại cấu kiện số có nhiều ưu điểm và cũng đã được phát triển rất mạnh. Về nguyên lý, chúng có cấu tạo rất giống với PROM. Việc lập trình cho SPLD có thể được thực hiện bằng các công nghệ khác nhau, dựa trên cơ sở thực hiện các kết nối bằng cách sử dụng cầu chì hoặc chuyển mạch. Một SPLD, được tạo thành bằng một số mảng cổng AND, OR, XOR hoặc cả các triger, có thể thực hiện nhiều hàm Boole khác nhau.

Các SPLD đều có cấu tạo dựa trên một trong hai dạng cấu trúc chính: mảng logic khả trình PLA (Programmable Logic Array) và logic mảng khả trình PAL (Programmable Array Logic).



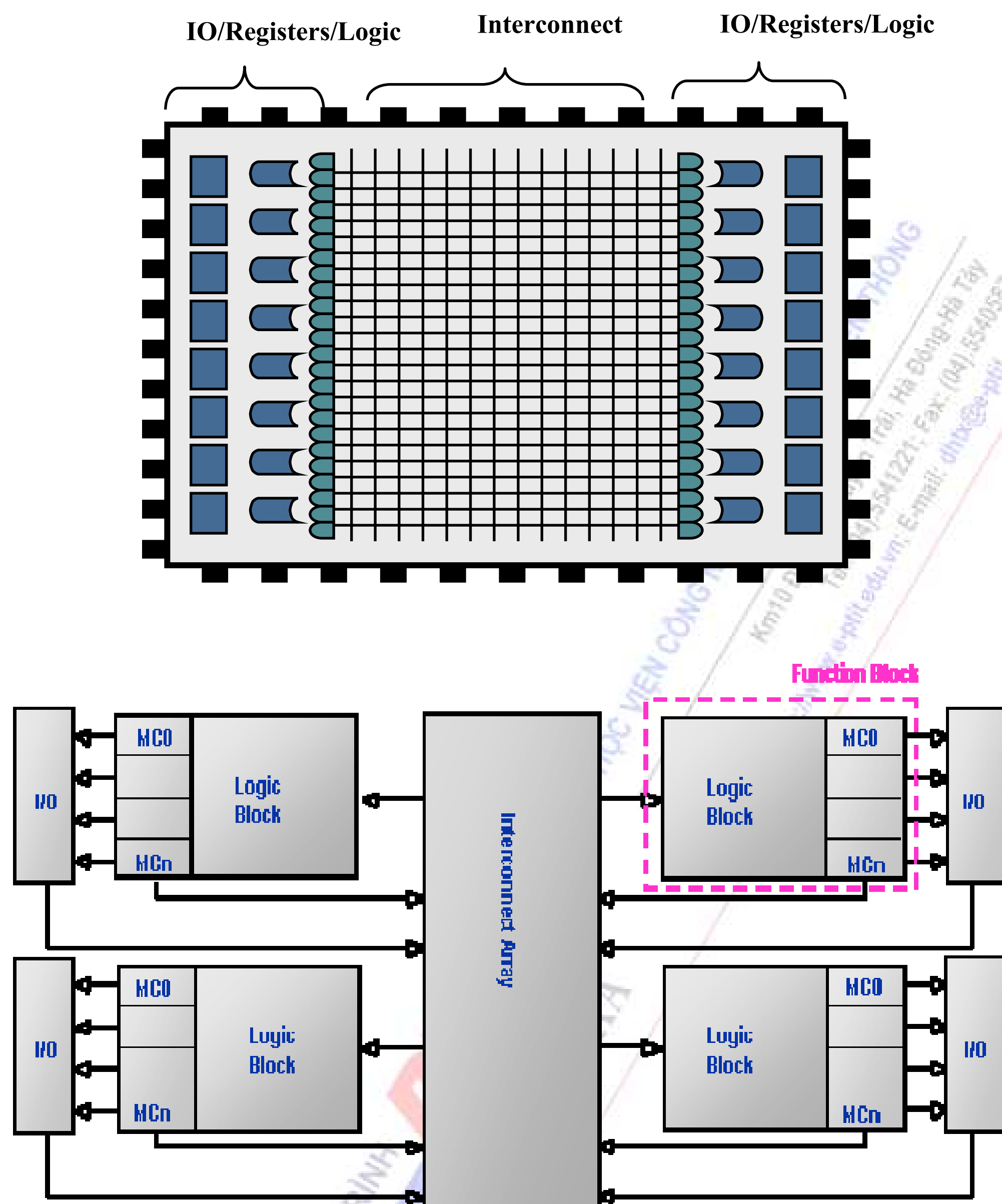
Hình 8.1 - So sánh giữa PAL và PLA

Thành phần cơ bản của PLA là một mảng AND và một mảng OR lập trình được. Mỗi mảng AND, OR gồm các hàng và các cột liên kết với nhau. Tại mỗi điểm giao giữa hàng và cột, có một cầu chì. Khi cầu chì đóng, tại điểm đó có kết nối giữa hàng và cột, khi cầu chì ngắt, tại đó không có kết nối. Việc đóng ngắt cầu chì được thực hiện bằng phần mềm (do lập trình viên hoặc sử dụng công cụ In- System Programming (ISP) – lập trình trên hệ thống).

Cấu trúc PLA tạo ra sự tổ hợp tùy ý giữa các cổng AND và OR, cho mật độ logic cao nhưng tốc độ chậm, số lượng cầu chì lớn. Vì vậy, sau này người ta đã đưa ra một kiểu kiến trúc khác là logic mảng khả trình PAL (Programmable Aray Logic).

Công nghệ PLD xuất hiện từ rất sớm với các công ty như Xilinx – sản xuất vi mạch CMOS công suất cực thấp dựa trên công nghệ Flash. PLD dựa trên công nghệ Flash cho phép lập trình và xoá vi mạch nhiều lần bằng điện, nhờ đó tiết kiệm được thời gian so với xoá vi mạch bằng tia cực tím.

8.3. CPLD (Complex PLD)



Hình vẽ 8.2 - Kiến trúc của CPLD

Thiết bị logic khả trình phức hợp (CPLD) có mật độ logic cao hơn so với các PLD đơn giản như đã xét ở trên (PLA và PAL). CPLD bao gồm nhiều mạch logic, mỗi mạch có thể coi là một SPLD. Trong một mạch đơn chỉ thực hiện các chức năng logic đơn giản. Các chức năng logic phức tạp hơn cần số lượng khối nhiều hơn, sử dụng ma trận liên kết chung giữa các khối để tạo kết nối. CPLD thường dùng để điều khiển ghép cổng phức hợp ở tốc độ rất cao (5ns, tương đương với 200 MHz). Kiến trúc cơ bản của CPLD được minh hoạ trong hình vẽ 8.2.

CPLD có cấu trúc đồng nhất gồm nhiều khối chức năng "**Function Block**" được kết nối với nhau thông qua một ma trận kết nối trung tâm "**Interconnect Array**". Mỗi khối **function block** gồm có một khối logic - gồm các hạng tích AND và OR sắp xếp giống PLA hoặc PAL, cho phép thực hiện các hàm logic tổ hợp- và nhiều khối MC (Macrocell) có chứa tài nguyên là các Trigơ cho phép xây dựng các thanh ghi và mạch tuần tự. Phần lõi bên trong của CPLD được nối ra bên ngoài thông qua các khối vào ra I/O cho phép thiết lập chức năng cho các chân của IC có chức

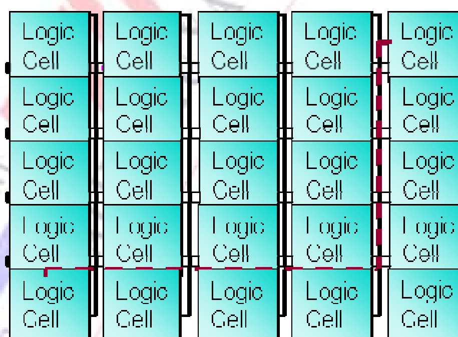
năng vào hoặc ra hoặc vừa là chân vào vừa là chân ra, ngoài ra còn có thể thiết lập các chân I/O này làm việc ở các mức logic khác nhau, có điện trở pull-up hoặc pull-down ...

Với cấu trúc đồng nhất, giá thành rẻ, tính năng khá mạnh, dễ sử dụng CPLD đã và đang được sử dụng rất rộng rãi trong thực tế, giúp cho nhà sản xuất phát triển nhanh sản phẩm của mình với giá thành rẻ. Đặc biệt hiện nay các hãng đã phát triển các họ CPLD với tính năng rất mạnh, công suất tiêu thụ thấp, chúng đang được sử dụng rất nhiều để phát triển các sản phẩm điện tử, viễn thông, công nghệ thông tin, nhất là trong các thiết bị cầm tay, di động…

Trong thực tế rất có nhiều loại CPLD khác nhau, của các hãng khác nhau, và đã được phát triển với nhiều chủng loại, thế hệ CPLD khác nhau. Cấu tạo, dung lượng, tính năng, đặc điểm, ứng dụng… của mỗi loại CPLD cũng rất khác nhau. Trong giáo trình này không đi sâu trình bày cấu tạo cụ thể của các họ CPLD, mà chỉ trình bày kiến trúc chung đơn giản nhất của CPLD. Khi sử dụng cụ thể loại CPLD nào, người học nên tham khảo các tài liệu khác, nhất là tham khảo các tài liệu kỹ thuật được cung cấp kèm theo cấu kiện do các hãng đưa ra. Các hãng điện tử nổi tiếng trên thế giới đang sở hữu, phát triển, cung cấp các loại cấu kiện CPLD là Xilinx, Altera…

8.4. FPGA

FPGA (Field Programmable Gate Array - Ma trận cổng lập trình được theo trường): có cấu trúc và hoạt động phức tạp hơn CPLD. Nó có thể thực hiện những chức năng phức tạp ưu việt hơn CPLD. Năm 1985, công ty Xilinx đưa ra ý tưởng hoàn toàn mới, đó là kết hợp thời gian hoàn thành sản phẩm và khả năng điều khiển được của PLD với mật độ và ưu thế về chi phí của GateArray. Từ đó, FPGA ra đời. Hiện nay, Xilinx vẫn là nhà sản xuất chip FPGA số một trên thế giới.

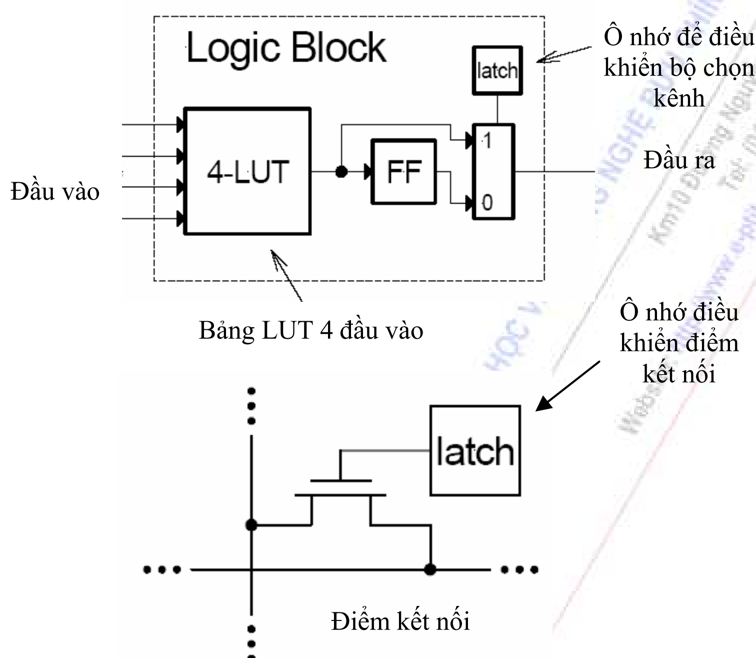


Hình 8-3. Kiến trúc chung của FPGA

Cấu trúc FPGA đơn giản gồm các tế bào logic (Logic Cell), các khối cách đều nhau, liên kết nhờ các đường kết nối có thể thay đổi được theo yêu cầu của người thiết kế. Nghĩa là người thiết kế có quyền thiết kế, lập trình và thay đổi mạch điện. Hiện nay, FPGA có mật độ khá cao, lên tới hàng trăm tỷ cổng và cấu trúc cũng đa dạng, phức tạp hơn. Nhiều chức năng phức tạp đã được tích hợp sẵn để tăng hiệu quả sử dụng FPGA. Ví dụ như ngoài những khối tế bào logic, nhiều họ FPGA đã được tích hợp thêm các khối chức năng như các bộ nhân cứng, khối nhớ, PLL, thậm chí cả một bộ vi xử lý mạnh…

Có hai loại FPGA cơ bản: loại lập trình lại được, dựa trên công nghệ SRAM và loại lập trình một lần.

- Loại lập trình lại được (dựa trên SRAM):
    - SRAM xác định các kết nối
    - SRAM định nghĩa các hàm logic trong bảng ánh xạ (LUT- Look Up Table)
- Loại lập trình một lần:
    - Kết nối dạng bẻ cầu chì
    - Sử dụng các cổng logic truyền thống



Hình 8.4 - Cấu trúc của logic cell đơn giản

Hai dạng này khác nhau về quy trình thực hiện tế bào logic và cơ chế được sử dụng để tạo kết nối trong thiết bị.

Chip FPGA lập trình một lần sử dụng phương pháp bẻ cầu chì (kết nối được tạo ra bằng cách đóng cầu chì) để tạo kết nối tạm thời trong chip, do đó không cần SPROM hoặc các phương tiện khác để nạp chương trình vào FPGA. Tuy nhiên, mỗi lần thay đổi thiết kế, phải bỏ hoàn toàn chip cũ đi. Tế bào logic OTP tương tự như PLD với các cổng và các trigơ định trước.

Dạng FPGA quan trọng hơn và được dùng phổ biến hơn cả là dạng lập trình lại được, dựa trên SRAM. Trên thực tế, FPGA SRAM được lập trình lại mỗi khi bật nguồn, vì FPGA là dạng chip nhớ tạm thời. Do đó, mỗi chip FPGA đều cần có một bộ nhớ PROM nối tiếp hoặc một bộ nhớ hệ thống.

Trong tế bào logic SRAM, thay vì các cổng thông thường, người ta sử dụng bảng ánh xạ (LUT). Bảng này xác định các giá trị đầu ra dựa trên các giá trị đầu vào, sử dụng để xây dựng các hàm logic tổ hợp. Trong sơ đồ "Tế bào logic SRAM" minh hoạ ở hình vẽ 8-3, 16 tổ hợp khác nhau của 4 đầu vào sẽ xác định giá trị của đầu ra). Các các ô nhớ SRAM cũng được sử dụng để điều khiển kết nối .

8.5. SO SÁNH GIỮA CPLD VÀ FPGA

CPLD	FPGA
- Cấu trúc theo mảng các hạng tích	- Cấu trúc dựa vào LUT
- Mảng kết nối trung tâm	- Ma trận kết nối 2 chiều X-Y
- Mật độ tích hợp trung bình	- Mật độ tích hợp cao
- Tỷ lệ số chân I/O trên microcell lớn	- Tỷ lệ số chân I/O trên microcell nhỏ
- Cấu hình được lưu lại khi mất điện, và không đổi trong quá trình hoạt động	- Cấu hình nạp vào SRAM, khi mất điện sẽ không còn, cần có bộ nhớ cấu hình PROM, cấu hình có thể được nạp động trong quá trình hoạt động.
- Cấu trúc đồng nhất	–Cấu trúc không đồng nhất
	– Nhiều tài nguyên: DLL (Delay_Locked Loop: Vòng khoá pha trễ), bộ nhớ, các bộ nhân
–Ứng dụng: mã hoá và giải mã logic, các máy trạng thái hay các giao diện bus chuẩn (SPI, I2C, SMBus...), ưu điểm nổi bật khi thiết kế các mạch logic nhiều đầu vào.	- Ứng dụng: PCI (Peripheral Component Interface), giao tiếp nối tiếp tốc độ cao và các bộ vi xử lý nhúng ...., ưu thế nổi bật khi thiết kế phức tạp, cần nhiều tài nguyên.

8.6. QUY TRÌNH THIẾT KẾ CHO CPLD/FPGA

Trong thực tế có rất nhiều hãng điện tử trên thế giới cung cấp các sản phẩm PLD và bộ công cụ phần mềm thiết kế đi kèm. Mỗi họ CPLD, FPGA của các hãng có những quy trình thiết kế khác nhau dành cho chúng, tuy nhiên về cơ bản chúng vẫn có quy trình thiết kế chung nhất định. Không mất tính tổng quát, để người học tiếp cận dễ dàng hơn, trong tài liệu này việc trình bày quy trình thiết kế cho CPLD/FPGA được lấy ví dụ, cụ thể hoá cho CPLD/FPGA của hãng Xilinx – Một hãng cung cấp các sản phẩm PLD số 1 thế giới hiện nay – sử dụng bộ công cụ phần mềm thiết kế ISE.

8.6.1. Yêu cầu chung khi thiết kế với CPLD/FPGA

8.6.1.1 Chọn vi mạch CPLD hoặc FPGA phù hợp

Khi phát triển các hệ thống số sử dụng CPLD/FPGA bước đầu tiên cần được thực hiện là phân tích bài toán, lựa chọn vi mạch CPLD hoặc FPGA phù hợp. Việc chọn được vi mạch, công nghệ phù hợp nhất cho các tiêu chuẩn thiết kế, được tiến hành theo các yêu cầu sau:

Mật độ: là mật độ logic dự tính của linh kiện, đặc trưng bởi khái niệm "số lượng cổng".

Số lượng thanh ghi: Phải tính được số thanh ghi cần cho bộ đếm, máy trạng thái, thanh ghi và bộ chốt. Số lượng macrocell trong vi mạch tối thiểu phải bằng số thanh ghi cần có.

Số lượng chân vào/ra: phải xác định vi mạch thiết kế cần bao nhiêu đầu vào, bao nhiêu đầu ra.

Yêu cầu về tốc độ: Tuyến tổ hợp nhanh nhất sẽ xác định tpd (trễ truyền trong vi mạch, tính theo ns). Mạch tuần tự nhanh nhất sẽ xác định tần số tối đa của vi mạch (f_{Max}).

Đóng vỏ: Phải xác định vi mạch cần gọn nhất hay chỉ sử dụng dạng QFP thông thường. Hoặc vi mạch thiết kế thuộc dạng có lắp chân cắm, trong trường hợp này là vi mạch PLCC.

Công suất thấp: Phải xác định sản phẩm sẽ sử dụng nguồn pin hay năng lượng mặt trời, thiết kế có yêu cầu công suất tiêu thụ thấp hay không, vấn đề tổn hao nhiệt có quan trọng hay không?

Chức năng cấp hệ thống: Phải xác định bo mạch có bao gồm nhiều vi mạch đa mức điện áp hay không, giữa các vi mạch có phải chuyển mức hay không, có yêu cầu sửa dạng xung đồng bộ hay không, có yêu cầu giao tiếp giữa bộ nhớ và bộ vi xử lý hay không?

8.6.1.2 Chọn giải pháp cấu hình cho CPDL/FPGA

Lập trình ngay trên hệ thống

Các CPLD và FPGA của các hãng nói chung, của Xilinx nói riêng có thể được lập trình ngay trên hệ thống (vi mạch đã được hàn vào mạch ứng dụng) thông qua giao thức JTAG (Joint Test Advisory Group: Chuẩn giao tiếp) đã được tích hợp sẵn trong IC. Người thiết kế sử dụng cáp nạp để nạp cấu hình cho CPLD hoặc FPGA. Xilinx đưa ra một chuẩn cáp nạp như sau:

+ **MultiLINX** : Cáp nạp dựa trên giao chuẩn giao tiếp nối tiếp USB hoặc RS232, cáp nạp này có tốc độ truyền trong dải rộng và giao diện có điện áp điều chỉnh được để phù hợp với việc giao tiếp với các hệ thống và các chân I/O hoạt động ở các mức điện áp khác nhau 5V; 3,3V; 2,5V. Và được thiết kế để hỗ trợ để cho các phần mềm gỡ rối phần cứng trước kia, nay chúng đã trở lên lỗi thời khi có sự ra đời của công cụ gỡ rối phần cứng ChipScope ILA.

+ **Parallel Cable IV:** Cáp nạp sử dụng cổng giao tiếp song song của máy tính, được phát triển để thay thế cho chuẩn cáp nạp Parallel Cable III và cho phép tăng tốc độ lên hơn 10 lần và hỗ trợ cho tất các các vi mạch sử dụng mức điện áp I/O từ 5V xuống 1,5V. Hiện nay chuẩn cáp nạp này được dùng phổ biến hơn cả.

Lập trình bên ngoài

Các CPLD và FPGA của Xilinx cũng có thể được lập trình bên ngoài bởi bộ lập trình chip HW130 của Xilinx cũng như các bộ lập trình của các nhà phát triển khác. Điều này cũng thuận tiện cho việc sử dụng các chip được lập trình trước trong thời gian sản xuất.

Cấu hình của CPLD được nạp vào FLASH nên khi mất điện cấu hình không bị mất đi, trong khi đó cấu hình khi hoạt động của FPGA được ghi vào SRAM nên sẽ mất đi khi mất điện, vì vậy cần sử dụng FPGA và kết hợp với PROM lưu cấu hình phù hợp, mỗi khi bật nguồn, cấu hình sẽ nạp tự động từ PROM vào FPGA. Có thể sử dụng PROM nối tiếp hoặc song song, tuy nhiên thì loại PROM nối tiếp hay được sử dụng hơn cả. Khi thiết kế cần chọn loại PROM có dung lượng phù hợp với mật độ của các loại FPGA khác nhau.

Ngoài ra Xilinx còn cung cấp các giải pháp được thiết kế trước, dễ sử dụng để cấu hình cho tất cả CPLD và FPGA của Xilinx, nhất là khi thiết kế các hệ thống phức tạp. Tất cả các nội dung liên quan đến cấu hình, PROM cho FPGA hay ISP cho CPLD, đều được đưa ra. Các giải pháp sử

dụng công cụ 3rd part boundary scan, các giải pháp phần mềm kèm theo, cáp ISP, thiết bị kiểm tra tự động ATE và hỗ trợ lập trình cũng như các thiết bị lưu trữ cấu hình.

Giải pháp cấu hình hiện đại nhất là nhóm cấu hình System ACE. Với giải pháp System ACE, người thiết kế có thể dễ dàng sử dụng giao diện vi xử lí trong System ACE để trực tiếp phối hợp cấu hình FPGA theo các yêu cầu của hệ thống. Giải pháp đầu tiên trong nhóm này là System ACE CF, cung cấp công nghệ điều khiển ổ đĩa Microdrive kích thước một inch và CompactFlash cũng như bộ lưu trữ cấu hình có dung lượng 8 gigabits. Ngoài ra, System ACE CF cũng được thiết kế trước, cung cấp các đặc tính hiện đại để tận dụng khả năng cấu hình lại linh hoạt của FPGA, bao gồm:

- Cấu hình multi-board từ một nguồn duy nhất
- Quản lí bitstream đa cấu hình
- Nâng cấp cấu hình qua mạng (IRL)
- Hot-swapping
- Khởi tạo trung tâm xử lí và lưu trữ phần mềm
- Mã hóa

Với System ACE CF, người thiết kế có thể thực hiện được gần như toàn bộ các yêu cầu cấu hình cho FPGA. Các khả năng bổ trợ hệ thống này cho phép người thiết kế sử dụng FPGA thỏa mãn các yêu cầu định trước về mặt thiết kế và thời gian xử lí lỗi. Ngoài ra, các cổng vi xử lí và cổng kiểm tra JTAG còn cho phép tích hợp System ACE trong mọi hệ thống.

Một số đặc điểm của giải pháp cấu hình System ACE:

- Độ linh hoạt: Với System ACE CF, có thể sử dụng một thiết kế cho nhiều ứng dụng khác nhau, nhờ đó giảm đáng kể thời gian hoàn thành sản phẩm. Thay vì thiết kế vài bo mạch tương tự nhau phù hợp với các chuẩn khác nhau, giờ đây người thiết kế chỉ phải thiết kế một bo mạch duy nhất với nhiều cấu hình được lưu trữ trong bộ nhớ System ACE CF. Mỗi bo có thể chọn các cấu hình phù hợp với các chuẩn khác nhau bằng cách khởi tạo giá trị mặc định tương ứng được lưu trong bộ nhớ ACE. Hệ thống còn cho phép lưu nhiều cấu hình cho một thiết kế trong một System ACE CF đơn. Ví dụ như trong quá trình thiết kế mẫu, người thiết kế có thể lưu các cấu hình hoạt động, cấu hình kiểm tra và cấu hình gỡ rối trong bộ nhớ ACE, đồng thời có thể chọn các cấu hình khác để chạy thử bản thiết kế của mình.

Để hỗ trợ quản lý nhiều bitstream và tích hợp điều khiển cấu hình FPGA với hoạt động của hệ thống, System ACE có một cổng vi xử lí trong hệ thống. Cổng này cho phép bộ xử lí của hệ thống thay đổi cấu hình mặc định, cấu hình lại trigơ, cấu hình lại từng FPGA hoặc một nhóm FPGA, truy nhập vào các file không cấu hình được lưu trong khối CompactFlash, hoặc dùng khối CompactFlash làm bộ nhớ chung cho hệ thống.

Với các FPGA có trung tâm xử lí kèm theo, System ACE CF cung cấp giải pháp 3 trong 1 để quản lí phần cứng và phần mềm. System ACE CF có thể cấu hình khung FPGA, khởi tạo trung tâm vi xử lí, và cung cấp các ứng dụng phần mềm cho trung tâm này nếu cần mà không phải thêm bất cứ thiết bị phần cứng nào.

+ ***Mật độ:*** Với mật độ logic cao chưa từng thấy (trên 8 Gb), một System ACE CF có thể cấu hình cho hàng trăm FPGA và có thể thay thế cho các mảng PROM cấu hình. Người thiết kế có thể lưu một số lượng lớn các thiết kế khác nhau cho một mảng FPGA trong cùng một khối nhớ. System ACE CF sử dụng hệ thống file FAT (File Allocation Table: bảng sắp xếp file tiêu chuẩn), do đó người thiết kế có thể lưu cả những file không ở dạng bitstream hoặc sử dụng bộ nhớ thừa làm bộ nhớ chuẩn cho hệ thống.

+ ***Khả năng quản lý tập trung:*** System ACE CF được thiết kế để quản lí cấu hình theo yêu cầu. Một System ACE CF có thể cấu hình cho một hoặc nhiều bo FPGA kết nối qua một back-plane. Khả năng tập trung cho phép đơn giản hóa quá trình quản lí và nâng cấp cấu hình. Để thay đổi hay nâng cấp cấu hình của một hệ thống, người thiết kế có thể vào khối nhớ, thực hiện các thay đổi cần thiết trên màn hình máy tính, chỉnh lại nội dung trong hệ thống qua cổng vi xử lí; hoặc tải cấu hình mới về qua mạng, sử dụng IRL.

8.6.1.3 Chọn công cụ phần mềm phù hợp

Xilinx đã cung cấp các công cụ thiết kế điện tử hoàn chỉnh, cho phép thực hiện thiết kế trên các thiết bị logic khả trình của Xilinx. Các công cụ này kết hợp công nghệ tiên tiến với giao diện đồ họa linh hoạt, dễ sử dụng để người thiết kế có được thiết kế tối ưu. Bộ công cụ phần mềm hiện đang được sử dụng rộng rãi là ISE với phiên bản mới nhất là 7.0 (năm 2005).

Xilink cũng cung cấp ISE dưới dạng các gói phần mềm có cấu hình khác nhau với giá thành khác nhau:

+ ISE WebPACK - bản miễn phí có thể dùng để thiết kế cho tất cả các họ CPLD của Xilinx

+ Gói phần mềm cơ bản BASEX: có thể thiết kế cho các loại chíp sau:

Virtex-4, FPGA LX15, LX25, SX25, FX12, Spartan-3 FPGA lên đến 1500 ngàn cổng và tất cả các họ CPLD.

+ Gói phần mềm Foundation: có thể thiết kế cho tất cả các loại FPGA và CPLD của Xilinx

Ngoài ra Xilinx còn phát triển các bộ công cụ phần mềm tiện ích khác như System Generator hỗ trợ cho các thiết kế DSP (Digital Signal Processor: Bộ xử lý tín hiệu số), hay EDK (Embbleded Dvelopment Kit: Bộ phần mềm phát triển hệ thống) hỗ trợ cho các thiết kế nhúng.

ISE được dùng kết hợp với phần mềm mô phỏng ModelSim của Mentor Graphics phiên bản XE được phát triển riêng hỗ trợ cho các họ CPLD/FPGA của Xilinx.

8.6.2 Lưu đồ thiết kế cho CPLD của Xilinx

Quá trình thiết kế cho CPLD chủ yếu là thực hiện trên các công cụ phần mềm, lưu đồ thiết kế chung cho CPLD (Ví dụ sử dụng phần mềm ISE) như hình vẽ sau, bao gồm các bước như sau:

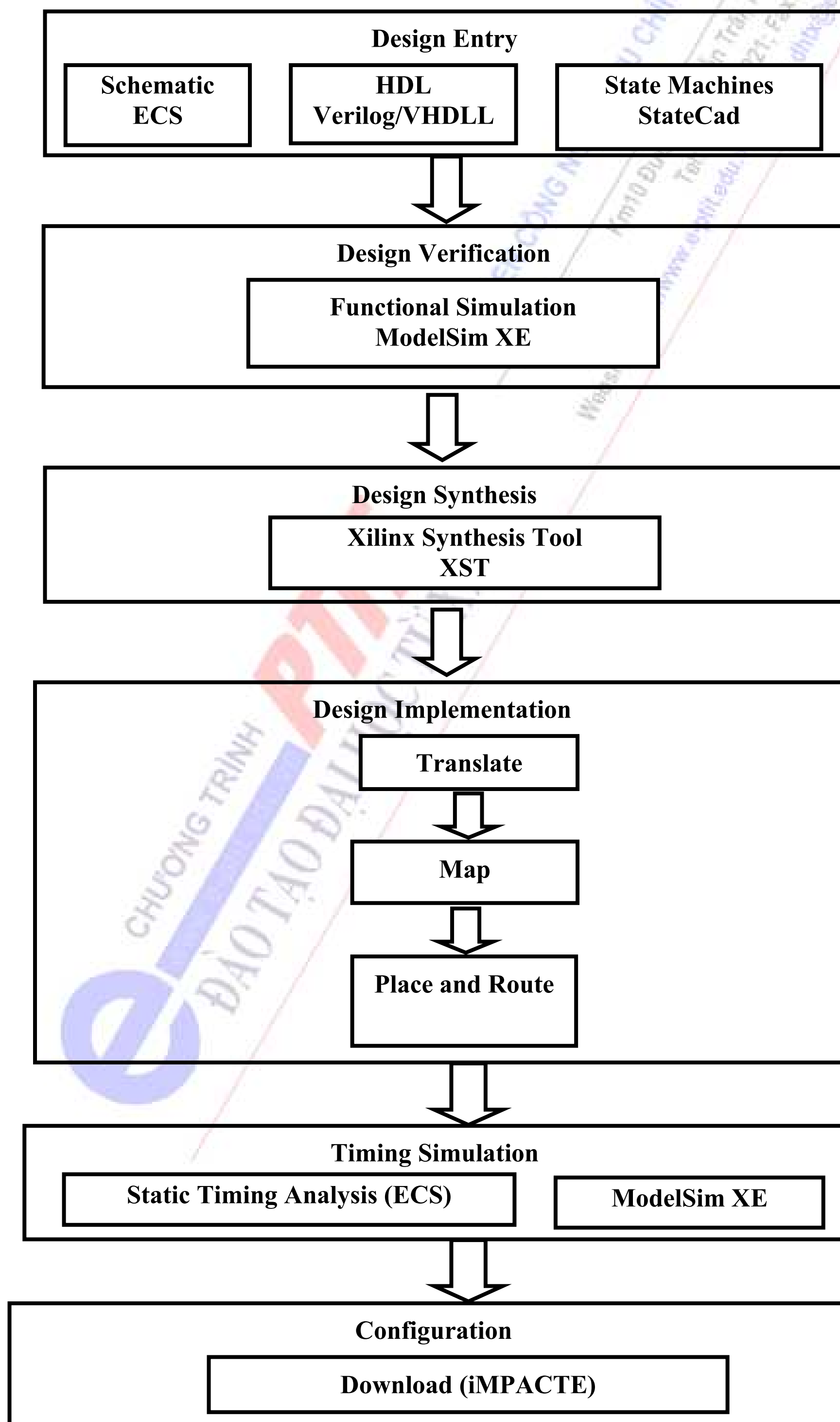
+ Nhập thiết kế (Design Entry):

Đây là bước đầu tiên và quan trọng nhất của quá trình thiết kế cho CPLD. Các công cụ thiết kế cho phép nhập thiết kế cho phép nhập thiết kế theo các cách sau:

- Nhập thiết kế theo sơ đồ nguyên lý Schematic, người thiết kế sử dụng các modul đã có sẵn trong thư viện Schematic để ghép nối chúng với nhau tạo thành bản thiết kế theo yêu cầu, cách này có thể thực hiện thiết kế nhanh nhưng sẽ rất khó khăn và không tối ưu tài nguyên của CPLD

khi thiết kế phức tạp, và thiết kế không thể sử dụng sang công cụ thiết kế CPLD của các hãng khác. Từ sơ đồ nguyên lý thiết kế được công cụ phần mềm sẽ chuyển đổi sang file ngôn ngữ mô tả phần cứng HDL, mà phổ biến là VHDL hoặc Verilog.

- Nhập thiết kế sử dụng ngôn ngữ mô tả phần cứng HDL (VHDL, Verilog, ABEL, AHDL...), Người thiết kế có thể sử dụng chương trình soạn thảo để thực hiện việc mô tả toàn bộ bản thiết kế của mình dưới dạng ngôn ngữ HDL nào đó mà công cụ thiết kế có thể tổng hợp được. Có rất nhiều phương pháp mô tả, mức độ trừu tượng khác nhau khi thiết kế, mỗi cách mô tả khác nhau có thể tạo ra một cấu trúc mạch khác nhau trong CPLD mặc dù chúng có cùng chức năng.



Hình 8.5- Lưu đồ thiết kế CPLD

Do đó người thiết kế cần thực hiện phân tích bài toán, tìm hiểu tài nguyên, cấu trúc của CPLD, yêu cầu về thời gian thiết kế để sử dụng kiểu mô tả. Mức độ trừu tượng trong khi mô tả phù hợp vừa đảm bảo yêu cầu về thời gian thiết kế vừa tối ưu được việc sử dụng tài nguyên của CPLD.

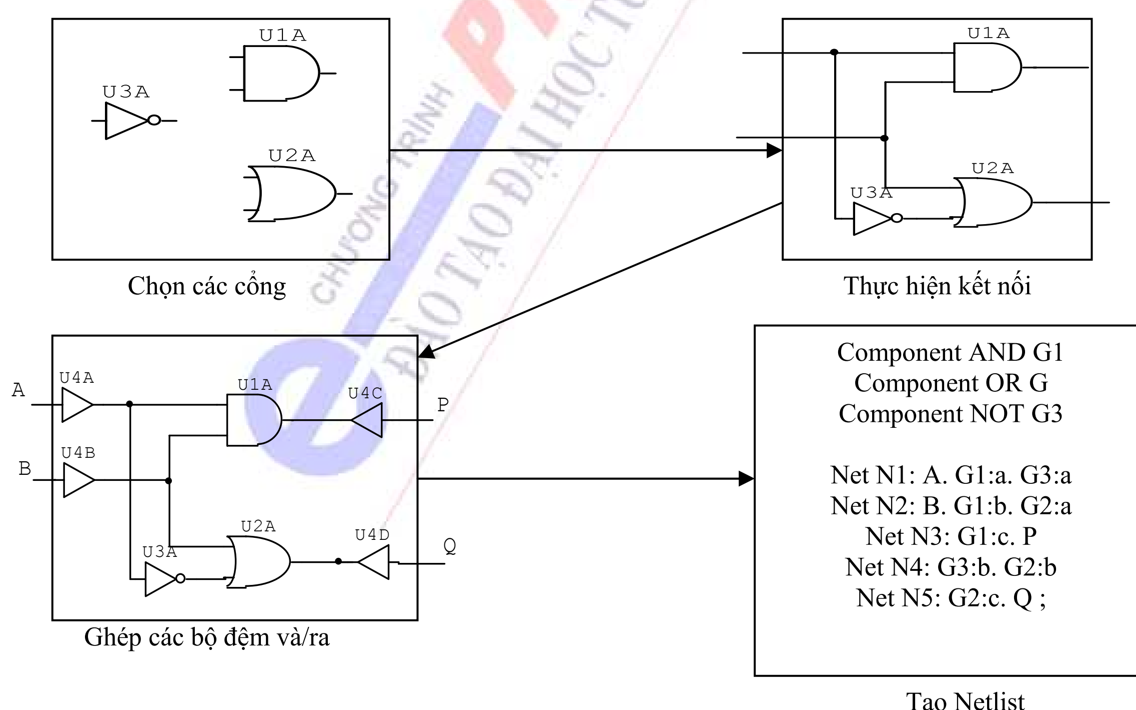
- Nhập thiết kế dưới dạng sơ đồ: Công cụ thiết kế còn cho phép nhập thiết kế vào dưới dạng sơ đồ mà điển hình là đồ hình trạng thái, sau đó chúng cũng được chuyển đổi sang HDL.

Việc nhập thiết kế rất linh hoạt, có thể sử dụng cả 3 cách trên để thực hiện các phần khác nhau của thiết kế.

+ ***Kiểm tra, mô phỏng thiết kế (Design Verification)***: Thực hiện kiểm tra, mô phỏng chức năng hoạt động của thiết kế HDL đã tạo ra ở trên. Các công cụ thiết kế đều hỗ trợ việc mô phỏng chức năng hoạt động của bản thiết kế HDL theo mô hình hoạt động (Behavioral Model), mức độ mô phỏng này độc lập với loại CPLD đã được lựa chọn. Bước này có thể không cần phải thực hiện trong khi thiết kế.

+ ***Tổng hợp thiết kế (Design Synthesis)***: Sau khi hoàn thành mô phỏng thiết kế, bước tổng hợp tiếp theo có nhiệm vụ chuyển thiết kế dưới dạng file vản bản HDL thành dạng file netlist, thực hiện mô tả mạch thực ở mức thấp dưới dạng cổng logic và kết nối giữa chúng với nhau. Có thể sử dụng các công cụ tổng hợp của các hãng khác nhau.

Mỗi công cụ có thể tạo ra file netlist theo định dạng riêng (ví dụ của XST của Xilinx XNF-Xilinx Netlist Format) nhưng có thể đặt lựa chọn để tạo ra file netlist dưới dạng định dạng chuẩn EDIF (Electronic Digital Interchange Format) mà tất cả các công cụ có thể hiểu được.



Hình 8.6- Ví dụ tổng hợp ra file netlist

+ ***Thực hiện thiết kế (Design Implementation)***: Sau khi có file netlist, bước tiếp theo là thực hiện thiết kế, nghĩa là xây dựng cấu hình cho CPLD. Bước này sử dụng file netlist và file ràng buộc "constraints File" (mô tả các nguyên tắc thiết kế, các ràng buộc về vật lý như gán vị trí cho các đầu vào/ra trên chip, các ràng buộc về tốc độ, thời gian, tần số…) để tạo thiết kế sử dụng tài nguyên có sẵn của CPLD. Bước này bao gồm các bước: Translate (biên dịch), Map (Phân bố bản thiết kế vào chip), Place and Route (Định vị và định tuyến kết nối).

+ ***Translate (biên dịch):*** Bước này nhằm thực hiện kiểm tra thiết kế và đảm bảo netlist phù hợp với kiến trúc đã chọn, kiểm tra file ràng buộc "constraints File" của người sử dụng để phát hiện các lỗi mâu thuẫn với tham số của chip đã chọn. Biên dịch thường bao gồm các quá trình: tối ưu hoá, biên dịch thành các thành phần vật lý của thiết bị; kiểm tra ràng buộc thiết kế. Khi kết thúc bước biên dịch, sẽ có một bản báo cáo về các chương trình được sử dụng, danh sách các cổng I/O và các thiết bị được sử dụng trong thiết kế, nhờ đó người thiết kế sẽ lựa chọn được phương án thiết kế tối ưu.

+ ***Map:*** tạo bản phân bố thiết kế tới các tài nguyên cụ thể trong CPLD. Nếu thiết kế quá lớn so với thiết bị được chọn, quy trình này không thể hoàn thành nhiệm vụ của mình. Quá trình Map có các tham số ràng buộc của thiết kế, ví dụ như tham số tốc độ, thời gian của thiết kế, và đôi khi quyết định gắn thêm các thành phần logic để đáp ứng các yêu cầu về thời gian. Map có khả năng thay đổi thiết kế xung quanh các bảng ánh xạ để tạo khả năng thực hiện tốt nhất cho thiết kế. Quy trình này được thực hiện hoàn toàn tự động và cần rất ít tác động đầu vào từ người sử dụng. Bước này nhằm đưa mạch thiết kế vào một thiết bị cụ thể. Bước này cũng tạo ra báo cáo xác nhận các tài nguyên được sử dụng trong chip, mô tả chính xác các phần trong thiết kế được đặt ở vị trí nào trong chip thực tế.

+ ***Place and Route*** (PAR - Định vị trí và định tuyến kết nối) Place là quá trình lựa chọn vị trí phù hợp của mỗi khối chức năng trong thiết kế và đưa các cổng logic của phần đó vào các khối logic hay các modul cụ thể trong CPLD trên cơ sở tối ưu việc kết nối và đảm bảo về các ràng buộc về thời gian. Những phần logic hoạt động tốc độ cao sẽ được xếp cạnh nhau để giảm độ dài đường kết nối. Route là quá trình tạo liên kết vật lý giữa các khối logic. Hầu hết các nhà sản xuất cung cấp công cụ Place and Route tự động cho người sử dụng. Ngoài công cụ tự động, người thiết kế có thể tự Place and Route trong khi thiết kế. Nhà sản xuất cũng cung cấp các công cụ, như “Floorplanner”, để nâng cao hiệu suất quá trình Place and Route do người thiết kế thực hiện so với quá trình tự động.

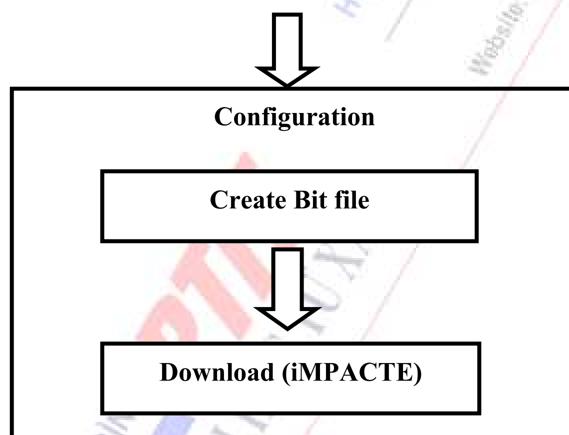
Place and Route là quá trình phức tạp, do đó nó chiếm thời gian nhiều nhất. Tuy nhiên, bước này chỉ có thể hoạt động tốt nếu chip đã chọn đáp ứng đủ các tuyến liên kết cho thiết kế. Nếu không, người thiết kế sẽ phải chọn chip có dung lượng lớn hơn. Sau bước này tạo ra được file cấu hình *.jed có thể được nạp vào cho CPLD.

+ ***Timing Simulation*** (***Mô phỏng có tham số thời gian)*****:** Sau bước Place and Route người thiết kế có thể thực hiện mô phỏng thiết kế ở mức cổng logic đã được định vị trí và định tuyến trên CPLD, phần mềm sử dụng file cấu hình đã được tạo ra và kết hợp với thư viện về mô hình thời gian của các họ CPLD (Ví dụ ISE của Xilinx thì dùng thư viện VITAL), để thực hiên mô phỏng hoạt động của thiết kế mà có tính đến các tham số thời gian trễ, thời gian thiết lập… của các cổng logic trong CPLD. Bước này rất quan trọng với những thiết kế phức tạp, tốc độ lớn.

+***Configuration (Cấu hình)***: Gọi chương trình điều khiển việc nạp cấu hình, thực kết nối thiết bị nạp (cáp nạp) đến CPLD và nạp file cấu hình cho CPLD. Với CPLD của hãng Xilinx, quá trình lập trình có thể thực hiện ngay trong hệ thống nhờ công cụ JTAG, hoặc sử dụng bộ lập trình thiết bị chuyên dùng, ví dụ như công cụ JTAG Data I/O, theo chuẩn IEEE/ ANSI 1149.1_1190. Công cụ JTAG là một bộ các nguyên tắc thiết kế, hỗ trợ quá trình kiểm tra, lập trình cho thiết bị và gỡ rối trên chip, trên bo mạch và trên hệ thống. Khả năng lập trình trên hệ thống là ưu điểm của CPLD, cho phép hàn trực tiếp thiết bị lên PCB. Nếu có thay đổi trong thiết kế, sẽ không phải tháo thiết bị ra khỏi bo mạch, mà đơn giản chỉ phải lập trình lại trên hệ thống.

8.6.3 Lưu đồ thiết kế cho FPGA

Lưu đồ thiết kế cho FPGA cũng tương tự như lưu đồ thiết kế cho CPLD, chỉ khác ở bước cuối cùng - bước Cấu hình cho FPGA. Ở bước này, đối với FPGA có thêm bước "Create Bit file" để tạo ra file "bitstream" để nạp vào bộ nhớ cấu hình trong FPGA thường là bộ nhớ tạm thời như SRAM. Dòng bit được nạp mang tất cả thông tin để định nghĩa các hàm logic và các liên kết trong thiết kế. Mỗi thiết kế khác nhau có một dòng bit khác nhau. Các thiết bị SRAM mất toàn bộ thông tin mỗi khi ngắt nguồn, do đó khi cần thiết phải nạp dòng bit cấu hình này vào trong PROM (thường sử dụng PROM nối tiếp). Mỗi khi thiết bị được bật nguồn file cấu hình từ PROM sẽ được nạp tự động vào bộ nhớ SRAM của FPGA, và FPGA hoạt động theo cấu hình đã được nạp đó.



Hình 8.6- Lưu đồ thiết kế FPGA

TÓM TẮT

Trong chương này trình bày các khái niệm cơ bản của logic lập trình. Với sự phát triển của các thiết bị logic lập trình ta có thể thiết kế các hệ thống kỹ thuật số phức tạp. Các kỹ thuật thiết kế ở cấp cao và các công cụ trợ giúp máy tính cần thiết để tạo nên chức năng thực thi PLD và FPGA hiệu quả. Việc thử nghiệm tính thực thi của PLD và FPGA cũng yêu cầu phải có các công cụ thử nghiệm và sự trợ giúp của máy tính.

CHƯƠNG 9: NGÔN NGỮ MÔ TẢ PHẦN CỨNG VHDL

GIỚI THIỆU

Trong toàn bộ lưu đồ thiết kế cho CPLD hoặc FPGA, bước nhập thiết kế là bước quan trọng và tốn nhiều công sức nhất, nó quyết định phần lớn đến kết quả của công việc thiết kế. Các công cụ thiết kế hỗ trợ nhiều phương pháp nhập thiết kế khác nhau, tuy nhiên phương pháp nhập thiết kế dùng ngôn ngữ mô tả phần cứng HDL là ưu việt hơn cả và được sử dụng chủ yếu trong quá trình thiết kế số nói chung và thiết kế cho CPLD/FPGA nói riêng. Hiện nay có nhiều ngôn ngữ HDL được sử dụng, tuy nhiên trong phần này chỉ giới thiệu phương pháp thiết kế dùng ngôn ngữ VHDL và giới thiệu những đặc điểm của VHDL khiến nó được trở thành một ngôn ngữ HDL đang được giảng dạy và sử dụng ở nhiều trường đại học trên thế giới.

Ngày nay, các mạch tích hợp ngày càng thực hiện được nhiều chức năng, do đó, vấn đề thiết kế mạch càng trở nên phức tạp. Những phương pháp truyền thống như dùng phương pháp tối thiểu hoá hàm Boolean hay dùng sơ đồ các phần tử không còn đáp ứng được các yêu cầu đặt ra khi thiết kế. Nhược điểm lớn nhất của các phương pháp này là chúng chỉ mô tả được hệ thống dưới dạng mạng nối các phần tử với nhau. Người thiết kế cần phải đi qua hai bước thực hiện hoàn toàn thủ công: đó là chuyển từ các yêu cầu về chức năng của hệ thống sang biểu diễn theo dạng hàm Boolean, sau các bước tối thiểu hoá hàm này ta lại phải chuyển từ hàm Boolean sang sơ đồ mạch của hệ thống. Cũng tương tự khi phân tích một hệ thống người phân tích cần phải phân tích sơ đồ mạch của hệ thống, rồi chuyển nó thành các hàm Boolean, sau đó mới lập lại các chức năng, hoạt động của hệ thống. Tất cả các bước nói trên hoàn toàn phải thực hiện thủ công không có bất kỳ sự trợ giúp nào của máy tính. Người thiết kế chỉ có thể sử dụng máy tính làm công cụ hỗ trợ trong việc vẽ sơ đồ mạch của hệ thống và chuyển từ sơ đồ mạch sang công cụ tổng hợp mạch vật lý dùng công cụ Synthesis. Một nhược điểm khác nữa của phương pháp thiết kế truyền thống là sự giới hạn về độ phức tạp của hệ thống được thiết kế. Phương pháp dùng hàm Boolean chỉ có thể dùng để thiết kế hệ thống lớn nhất biểu diễn bởi vài trăm hàm. Còn phương pháp dựa trên sơ đồ chỉ có thể dùng để thiết kế hệ thống lớn nhất chứa khoảng vài nghìn phần tử.

Phương pháp thiết kế, thử nghiệm, phân tích các hệ thống số sử dụng các ngôn ngữ mô tả phần cứng nổi bật lên với các ưu điểm hơn hẳn và sẽ dần thay thế các phương pháp truyền thống. Sự ra đời của ngôn ngữ mô phỏng phần cứng đã giải quyết được rất nhiều nhược điểm lớn của các phương pháp thiết kế trước đây: Nếu các phương pháp cũ đòi hỏi phải chuyển đổi từ mô tả hệ thống (các chỉ tiêu về chức năng) sang tập hợp các hàm logic bằng tay thì bước chuyển đó hoàn toàn không cần thiết khi dùng HDL. Hầu hết các công cụ thiết kế dùng ngôn ngữ mô phỏng phần cứng đều cho phép sử dụng biểu đồ trạng thái (finite-state-machine) cho các hệ thống tuần tự cũng như cho phép sử dụng bảng chân lý cho hệ thống tổng hợp. Việc chuyển đổi từ các biểu đồ trạng thái và bảng chân lý sang mã ngôn ngữ mô phỏng phần cứng được thực hiện hoàn toàn tự động.

Nhờ tính dễ kiểm tra thử nghiệm hệ thống trong suốt quá trình thiết kế mà người thiết kế có thể dễ dàng phát hiện các lỗi thiết kế ngay từ những giai đoạn đầu, giai đoạn chưa đưa vào sản xuất thử, do đó tiết kiệm được lượng chi phí đáng kể bởi từ ý thiết kế đến tạo ra sản phẩm đúng như mong muốn là một việc rất khó tránh khỏi những khó khăn, thất bại.

Khi mọi lĩnh vực của khoa học đều phát triển không ngừng thì sự phức tạp của hệ thống điện tử cũng ngày một tăng theo và gần như không thể tiến hành thiết kế thủ công mà không có sự trợ giúp cuả các loại máy tính hiện đại. Ngày nay, ngôn ngữ mô tả phần cứng HDL được dùng nhiều để thiết kế cho các thiết bị logic lập trình được PLD từ loại đơn giản đến các loại phức tạp như FPGA.

NỘI DUNG

9.1. GIỚI THIỆU NGÔN NGỮ MÔ TẢ PHẦN CỨNG VHDL

VHDL là ngôn ngữ mô tả phần cứng cho các mạch tích hợp tốc độ rất cao, là một loại ngôn ngữ mô tả phần cứng được phát triển dùng cho trương trình VHSIC(Very High Speed Itergrated Circuit) của bộ quốc phòng Mỹ. Mục tiêu của việc phát triển VHDL là có được một ngôn ngữ mô phỏng phần cứng tiêu chuẩn và thống nhất cho phép thử nghiệm các hệ thống số nhanh hơn cũng như cho phép dễ dàng đưa các hệ thống đó vào ứng dụng trong thực tế. Ngôn ngữ VHDL được ba công ty Intermetics, IBM và Texas Instruments bắt đầu nghiên cứu phát triển vào tháng 7 năm 1983. Phiên bản đầu tiên được công bố vào tháng 8-1985. Sau đó VHDL được đề xuất để tổ chức IEEE xem xét thành một tiêu chuẩn chung. Năm 1987 đã đưa ra tiêu chuẩn về VHDL(tiêu chuẩn IEEE-1076-1987).

VHDL được phát triển để giải quyết các khó khăn trong việc phát triển, thay đổi và lập tài liệu cho các hệ thống số. Như ta đã biết, một hệ thống số có rất nhiều tài liệu mô tả. Để có thể vận hành bảo trì sửa chữa một hệ thống ta cần tìm hiểu kỹ lưỡng tài liệu đó. Với một ngôn ngữ mô phỏng phần cứng tốt việc xem xét các tài liệu mô tả trở nên dễ dàng hơn vì bộ tài liệu đó có thể được thực thi để mô phỏng hoạt động của hệ thống. Như thế ta có thể xem xét toàn bộ các phần tử của hệ thống hoạt động trong một mô hình thống nhất.

VHDL được phát triển như một ngôn ngữ độc lập không gắn với bất kỳ một phương pháp thiết kế, một bộ mô tả hay công nghệ phần cứng nào. Người thiết kế có thể tự do lựa chọn công nghệ, phương pháp thiết kế trong khi chỉ sử dụng một ngôn ngữ duy nhất. Và khi đem so sánh với các ngôn ngữ mô phỏng phần cứng khác đã kể ra ở trên ta thấy VHDL có một số ưu điểm hơn hẳn các ngôn ngữ khác:

+ Thứ nhất là tính công cộng: VHDL được phát triển dưới sự bảo trợ của chính phủ Mỹ và hiện nay là một tiêu chuẩn của IEEE. VHDL được sự hỗ trợ của nhiều nhà sản xuất thiết bị cũng như nhiều nhà cung cấp công cụ thiết kế mô phỏng hệ thống.

+ Thứ hai là khả năng hỗ trợ nhiều công nghệ và phương pháp thiết kế. VHDL cho phép thiết kế bằng nhiều phương pháp ví dụ phương pháp thiết kế từ trên xuống, hay từ dưới lên dựa vào các thư viện sẵn có. VHDL cũng hỗ trợ cho nhiều loại công cụ xây dựng mạch như sử dụng công nghệ đồng bộ hay không đồng bộ, sử dụng ma trận lập trình được hay sử dụng mảng ngẫu nhiên.

+ Thứ ba là tính độc lập với công nghệ: VHDL hoàn toàn độc lập với công nghệ chế tạo phần cứng. Một mô tả hệ thống dùng VHDL thiết kế ở mức cổng có thể được chuyển thành các bản tổng hợp mạch khác nhau tuỳ thuộc công nghệ chế tạo phần cứng mới ra đời nó có thể được áp dụng ngay cho các hệ thống đã thiết kế .

+ Thứ tư là khả năng mô tả mở rộng: VHDL cho phép mô tả hoạt động của phần cứng từ mức hệ thống số cho đến mức cổng. VHDL có khả năng mô tả hoạt động của hệ thống trên nhiều mức nhưng chỉ sử dụng một cú pháp chặt chẽ thống nhất cho mọi mức. Như thế ta có thể mô phỏng một bản thiết kế bao gồm cả các hệ con được mô tả chi tiết.

+ Thứ năm là khả năng trao đổi kết quả: Vì VHDL là một tiêu chuẩn được chấp nhận, nên một mô hình VHDL có thể chạy trên mọi bộ mô tả đáp ứng được tiêu chuẩn VHDL. Các kết quả mô tả hệ thống có thể được trao đổi giữa các nhà thiết kế sử dụng công cụ thiết kế khác nhau nhưng cùng tuân theo tiêu chuẩn VHDL. Cũng như một nhóm thiết kế có thể trao đổi mô tả mức cao của các hệ thống con trong một hệ thống lớn (trong đó các hệ con đó được thiết kế độc lập).

+ Thứ sáu là khả năng hỗ trợ thiết kế mức lớn và khả năng sử dụng lại các thiết kế: VHDL được phát triển như một ngôn ngữ lập trình bậc cao, vì vậy nó có thể được sử dụng để thiết kế một hệ thống lớn với sự tham gia của một nhóm nhiều người. Bên trong ngôn ngữ VHDL có nhiều tính năng hỗ trợ việc quản lý, thử nghiệm và chia sẻ thiết kế. Và nó cũng cho phép dùng lại các phần đã có sẵn.

9.2. CẤU TRÚC NGÔN NGỮ CỦA VHDL

VHDL là ngôn ngữ cho phép mô tả các thiết bị phần cứng số trừu tượng, nó không dựa vào công nghệ thiết bị phần cứng số, phương pháp được sử dụng để thiết kế thiết bị số, mà những khái niệm, mô hình trừu tượng của thiết bị phần cứng số được đưa ra như là nền tảng của ngôn ngữ. Do đó dùng VHDL cho phép mô tả được hầu hết các hệ thống phần cứng số. Các mô hình trừu tượng gồm:

- Mô hình hoạt động (a Model of Behavior).
- Mô hình thời gian (a Model of Time).
- Mô hình cấu trúc (a Model of Structure).

Để thực hiện mô tả cho một hệ thống số nào đó cần thực hiện theo các bước như sau:

+ Phân tích yêu cầu của hệ thống số cần phải thiết kế hoặc cần phải mô tả.

+ Phân tách hệ thống thành những khối con.

+ Xác định mô hình mô tả phù hợp cho mỗi khối con hoặc cho cả hệ thống.

+ Sử dụng ngôn ngữ VHDL để mô tả hệ thống số theo các mô hình đã xác định.

Như vậy việc nắm chắc cấu trúc, cú pháp, các mô hình mô tả của ngôn ngữ là rất quan trọng, quyết định chủ yếu đến thành công trong việc mô tả hệ thống số cần thiết kế.

VHDL cũng có nhiều điểm giống như một ngôn ngữ lập trình bậc cao, có cấu trúc, có cú pháp riêng, có cách tổ chức chương trình, có từ khóa, có phương pháp biểu diễn số liệu riêng...

<u>Chú ý</u>: - Trong các đoạn mã mô tả VHDL trong chương các từ khóa đều được in đậm.

- Trong VHDL không phân biệt chữ hoa, chữ thường.

9.2.1 Đối tượng trong VHDL

Trong ngôn ngữ VHDL gồm có 3 đối tượng là: tín hiệu **- signal**, biến **- variable**, hằng - **constant**, mỗi đối tượng được khai báo dựa vào từ khóa tương ứng và chúng có mục đích sử dụng như sau:

+ Tín hiệu – Signal: là đối tượng để biểu diễn đường kết nối các giữa các cổng vào/ra của thực thể, giữa các cổng vào/ra của các khối thành phần phần cứng xuất hiện trong thực thể… Chúng là phương tiện truyền dữ liệu động giữa các thành phần của thực thể.

Tín hiệu có tính toàn cục rất cao, chúng có thể được khai báo trong package (tín hiệu toàn cục, được sử dụng bởi một số thực thể), khai báo trong thực thể - Entity (tín hiệu nội bộ dùng trong thực thể, có thể được tham chiếu bởi bất kỳ kiến trúc nào của thực thể đó), khai báo trong kiến trúc – Architecture (tín hiệu nội bộ dùng trong kiến trúc, có thể được sử dụng trong bất cứ cấu trúc lệnh nào trong kiến trúc). Các tín hiệu có thể được sử dụng nhưng không được khai báo trong tiến trình – process, trong chương trình con. Vì tiến trình và chương trình con là thành phần cơ sở của mô hình và chúng được coi như các hộp đen. Cú pháp khai báo tín hiệu như sau:

```
   Signal tên_tín_hiệu {,tên_tín_hiệu}:kiểu_dữ_liệu [:=giá_trị_khởi_tạo];
Ví dụ: Signal a,b,c: Bit:='1';
       Signal y, reg: std_logic_vector(3 downto 0):="0000";
```

+ Biến – Variable: là đối tượng cục bộ được sử dụng để chứa các kết quả trung gian. Biến chỉ được khai báo và sử dụng trong process và trong chương trình con. Cú pháp khai báo của biến cũng tương tự như khai báo tín hiệu:

```
     variable tên_biến {,tên_biến}: kiểu_dữ_liệu [:=giá_trị_khởi_tạo];
Ví dụ: variable x : Bit:='1';
       variable Q: std_logic_vector(3 downto 0);
```

Nếu không được khởi tạo giá trị ban đầu biến sẽ nhận giá trị khởi tạo ban đầu là giá trị thấp nhất trong các giá trị thuộc miền xác định của kiểu dữ liệu. Tín hiệu cũng có thể chứa dữ liệu nhưng chúng lại không được sử dụng vì những lý do sau:

- Việc sử dụng biến hiệu quả hơn vì giá trị của biến được gán ngay lập tức trong process khi tín hiện chỉ được lập kế hoạch để thực hiện và chỉ được cập nhật toàn bộ sau khi kết thúc process.
- Biến chiếm ít bộ nhớ hơn trong khi tín hiệu cần nhiều thông tin để có thể lập kế hoạch thực hiện cũng như để chứa các thuộc tính của tín hiệu.
- Sử dụng tín hiệu yêu cầu có lệnh **wait** để thực hiện đồng bộ phép gán tín hiệu với phép lặp thực hiện theo cách sử dụng quen thuộc.

+ Hằng –constant: là đối tượng hằng được gán cho các giá trị cụ thể của một kiểu khi được tạo ra và không đổi trong toàn bộ quá trình thực hiện. Hằng cũng có tính toàn cục giống như tín hiệu và có thể được khai báo trong package, entity, architecture, proceduce, process… Cú pháp khai báo hằng:

```
          constant tên_hằng {,tên_hằng}: kiểu_dữ_liệu :=giá_trị_khởi_tạo;
Ví dụ: constant GND : std_logic:='0';
       constant PI: real:=3.1414;
```

Tóm lại: Các đối tượng trong VHDL có mục đích sử dụng, phạm vi sử dụng khác nhau, nhưng chúng có cú pháp khai báo chung như sau:

```
Đối_tượng tên_đối_tượng : kiểu_dữ_liệu {:=giá_trị_khởi_tạo}
```

Các đối tượng khi khai báo phải được xác định kiểu dữ liệu tương ứng. VDHL định nghĩa nhiều kiểu dữ liệu khác nhau để phù hợp với việc mô tả, thiết kế, mô phỏng các hệ thống số khác nhau trong thực tế.

9.2.2 Kiểu dữ liệu trong VHDL

Trong VHDL có 4 dạng dữ liệu:

- Vô hướng : gồm các dữ liệu có giá trị đơn như bit, boolean, integer, real, physical, character, std_logic và std_ulogic, enumerated (kiểu liệt kê)...
- Kiểu ghép: các dữ liệu dưới dạng một nhóm các thành phần như mảng, bảng ghi (record). Bit_logic_vector, std_logic_vector và String đều là những dạng dữ liệu ghép đã được định nghĩa sẵn.
- 2-D Arrays: các dữ liệu có dạng mảng 2 chiều, được tạo nên từ 1 mảng của một mảng 1 chiều (hay một bản ghi).
- VHDL Subtypes: dạng dữ liệu con do người dùng tự định nghĩa dựa trên những dạng có sẵn.

Các kiểu dữ liệu đã được định nghĩa trong gói Standard chứa trong thư viện chuẩn Standard Library của VHDL là: bit, boolean, integer, real, physical, character, std_logic and std_ulogic, Bit_logic_vector, std_logic_vector và String và một số kiểu dữ liệu con. Cú pháp chung định nghĩa kiểu dữ liệu như sau:

```
Type Tên_kiểu is giới_hạn_giá_trị_của_kiểu
```

a. Kiểu vô hướng

- Kiểu Bit : Kiểu liệt kê với 2 giá trị ‘0’ và ‘1’. Kiểu Bit đã được định nghĩa như sau: `Type Bit is (‘0’, ‘1’);`

- Kiểu Boolean: Kiểu liệt kê với 2 giá trị false và true. Kiểu Boolean đã được định nghĩa như sau: **Type** Boolean **is** (false, true);

- Kiểu Integer: Kiểu số nguyên với những giá trị dương hoặc âm, độ lớn mặc định là 32 bit với giới hạn giá trị: từ -2147483647 đến +2147483647. Khi sử dụng có thể giới hạn miền xác định theo giới hạn giảm dần dùng từ khóa **downto** hoặc tăng dần dùng từ khóa **to**:

```
signal A : integer range 0 to 7; -- A số nguyên 3 bit
variable B : integer range 15 downto 0; -- B số nguyên 4 bit
signal B : integer range 15 downto -15; -- B số nguyên 5 bit
```

Các cách biểu diễn số nguyên dạng thập phân:

+ digit[underline]digit, ví dụ : 0, 1, 123_456_789 , -123_5678…

+ digit(**E**)digit, ví dụ: 987E6 (=987.10^6)…

Các cách biểu diễn dưới dạng cơ số xác định:

+ base#based_integer#[exponent], ví dụ: 2#1100_0100#, 16#C4#, 4#301#E1, (=196)

- Kiểu Real: Kiểu số thực có giới hạn từ -1.0E+38 đến 1.0E+38, khác với kiểu integer kiểu số thực khi sử dụng thường được định nghĩa thành kiểu dữ liệu riêng và có giới hạn miền xác định:

```
signal a: Real:=-123E-4;
type CAPACITY is range -25.0 to 25.0 ;
signal  Sig_1 : CAPACITY := 3.0 ;
type PROBABILITY is range 1.0 downto 0.0;
constant  P : PROBABILITY := 0.5 ;
```

Các cách biểu diễn số thực:

+ Biểu diễn dưới dạng thập phân: integer[.integer][exponent], ví dụ: 0.0, 0.5, 1.1234_5678, 12.4E-9…

+ Biểu diễn dưới dạng cơ số xác định:

base#based_integer[.based_integer]#[exponent]

Ví dụ: 2#1.111_1111_111#E+11, 16#F.FF#E2 (=4095.0)

- Kiểu Character: Kiểu kiểu ký tự, liệt kê với miền xác định là tập hợp các ký tự ASCII. Biểu diễn của giá trị Character: ‘A’, ‘a’, ‘*’, ‘ ‘, NUL, ESC…

- **Kiểu Vật lý – Physical**: được sử dụng để biểu diễn các đại lượng vật lý như khoảng cách, điện trở, dòng điện, thời gian… Kiểu vật lý cung cấp đơn vi cơ bản và các đơn vị kế tiếp được định nghĩa theo đơn vị cơ bản, đơn vị nhỏ nhất có thể biểu diễn được là đơn vị cơ bản. Trong thực việc chuẩn Time (kiểu dữ liệu thời gian) là kiểu vật lý duy nhất đã được định nghĩa.

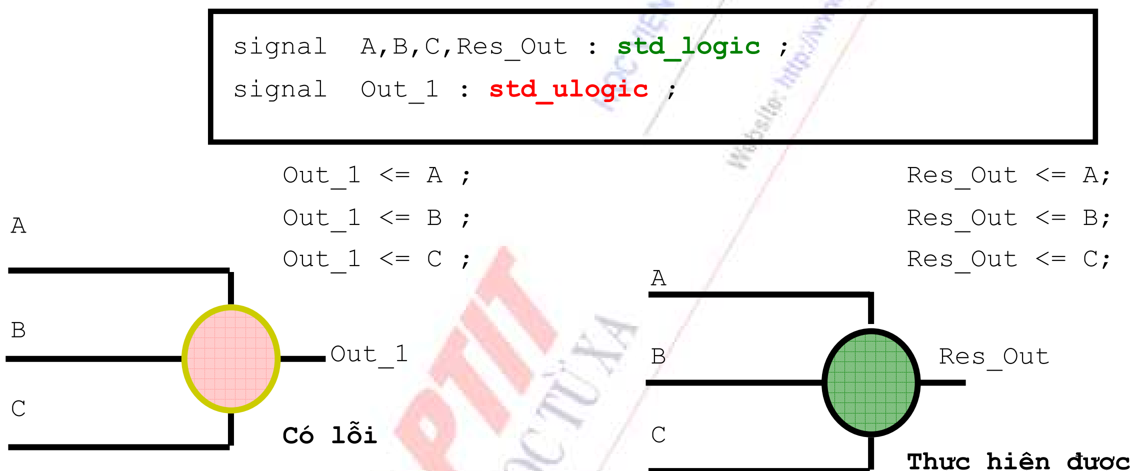
```
type  Time is range <xác_định giới hạn>
units
   fs; -- Đơn vị cơ bản
   ps = 1000 fs;
   ns = 1000 ps;
   us = 1000 ns;
   ms = 1000 us;
   sec = 1000 ms;
   min = 60 sec;
   hr = 60 min;
End Units;
```

```
Ví dụ sử dụng:
constant Tpd : time := 3ns ;
...
Z <= A after Tpd ;
```

- Kiểu std_logic và std_ulogic: kiểu dữ liệu logic nhiều mức đã được định nghĩa trong gói std_logic_1164, so với kiểu Bit thì chúng có thể mô tả chính xác và chi tiết hơn cho các phần cứng số, chúng còn xác định được cường độ khác nhau của các tín hiệu.

type std_ulogic **is**	**type** std_logic **is**
('U', -- Uninitialize	('U', -- Uninitialize
X', -- Forcing Unknown	X', -- Forcing Unknown
'0', -- Forcing Zero	'0', -- Forcing Zero
'1', -- Forcing One	'1', -- Forcing One
'Z', -- High Impedance	'Z', -- High Impedance
'W', -- Weak Unknown	'W', -- Weak Unknown
'L', -- Weak Zero	'L', -- Weak Zero
'H', -- Weak One	'H', -- Weak One
'-' -- Don't Care	'-' -- Don't Care
) ;	) ;

Hai kiểu dữ liệu std_logic và std_ulogic tương tự nhau, chúng chỉ khác nhau ở chỗ là kiểu std_ulogic không có hàm phân dải (unresolved) – hàm quyết định giá trị tín hiệu, do đó sẽ có lỗi khi các tín hiệu kiểu std_ulogic được nối chung vào 1 điểm. Thư viện cũng cung cấp hàm phát hiện lỗi này của các tín hiệu kiểu std_ulogic.



(Ký hiệu "<=" dùng ở trên là lệnh gán tín hiệu, lệnh gán tín hiệu thực hiện được với 2 dữ liệu cùng kiểu, cùng độ lớn, giá trị của tín hiệu bên phải sẽ được gán cho tín hiệu bên trái).

- Kiểu dữ liệu liệt kê tự định nghĩa: Kiểu dữ liệu liệt kê, do người sử dụng tự định nghĩa, cho phép mô tả rất sáng sủa, và linh hoạt cho các mô hình phần cứng số với mức độ trừu tượng cao. Kiểu dữ liệu này dùng nhiều mô tả đồ hình trạng thái, các hệ thống phức tạp…

Ví dụ: **type** My_State **is**(RST, LOAD, FETCH, STOR, SHIFT) ;

```
. . .
signal  STATE, NEXT_STATE : My_State ;
```

b. Kiểu dữ liệu ghép

Tương tự các ngôn ngữ lập trình, VHDL cũng có các kiểu dữ liệu ghép là nhóm các phần tử dữ liệu theo dạng mảng (array) hoặc bảng ghi (record).

+ Mảng – Array:

Mảng là nhóm nhiều phần tử có cùng kiểu dữ liệu với nhau thành đối tượng duy nhất. Mỗi phần tử của mảng có thể được truy cập bằng một hoặc nhiều chỉ số của mảng. Cú pháp định nghĩa kiểu dữ liệu mảng như sau:

```
Type tên_mảng is array (khoảng _của _chỉ số) of kiểu_của_phần_tử;
```

Ví dụ một số cách khai báo và sử dụng dữ liệu mảng:

```
type WORD is array (3 downto 0) of std_logic ;
signal B_bus : WORD ;
type DATA is array (3 downto 0) of integer range 0 to 9   ;
signal C_bus : DATA ;
```

Các kiểu dữ liệu mảng đã được định nghĩa trong thư viện chuẩn của VHDL là: Bit_logic_vector (mảng dữ liệu kiểu Bit), std_logic_vector (mảng dữ liệu kiểu std_logic) và String (mảng dữ liệu kiểu Chacracter). Một số ví dụ sử dụng các kiểu dữ liệu này như sau:

```
signal My_BusA, My_BusB: bit_vector (3 downto 0);
signal My_BusC : bit_vector (0 to 3) ;
signal Data_Word : std_logic_vector (11 downto 0);
variable Warning2: string(1 to 30):= " Unstable, Aborting Now" ;
constant Warning3: string(1 to 20):= " Entering FSM  State2";
```

Một số phép toán thao tác với phần tử mảng:

- Phép gán cho mảng: 2 mảng phải cùng kiểu, cùng độ lớn, phép gán sẽ thực hiện gán theo từng phần tử theo thứ tự từ trái sang phải:




```
Data_Word <= "101001101111" ;
Data_Word <= X"A6F";
Data_Word <= O"5157";
Data_Word <= B"1010_0110_1111" ;
```

Cách biểu diễn số liệu bit_vector và std_logic_vector: B|O|X "giá_trị" (dùng dấu nháy kép). Trong đó B : Binary -Kiểu nhị phân, O: Octal - kiểu bát phân, X: hexadecimal.

```
X"1AF"=B"0001_1010_1111"= B"000_110_101_111"=O"0657"
```

- Phép gộp ( ): cho phép nhóm cả dữ liệu vô hướng và dữ liệu mảng để thuận tiện cho các phép gán cho mảng:

```
signal H_BYTE, L_BYTE: std_logic_vector ( 0 to 7);
signal Q_Out : std_logic_vector (31 downto 0);
signal A, B, C, D  : std_logic;
signal WORD : std_logic_vector (3 downto 0);
...
(A,B,C,D)<=WORD;
```

Chú ý: Phép gộp ở vế bên trái chỉ dùng với kiểu dữ liệu vô hướng.

```
WORD <= ( 2 => '1', 3 => D, others => '0' ) ;
Q_Out <= (others => '0') ;
WORD <= ( A, B, C, D ) ;
H_Byte <= (7|6|0=>'1', 2 to 5 => '0' );
L_Byte <= (3=>'1', 1 to 2 => '0', 4 to 7 => '1');
```

Chú ý: "**others**" có thể được sử dụng khi gán mặc định, nó có ý nghĩa là các tất cả các phần tử còn lại được gán bằng một giá trị nào đó) .

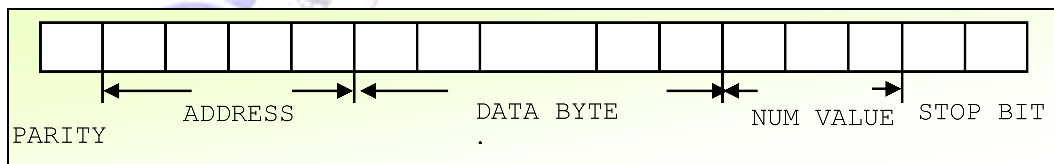
+ Bảng ghi – Record:

Bảng ghi là nhóm nhiều phần tử có kiểu dữ liệu khác nhau thành đối tượng duy nhất.

- Mỗi phần tử của bản ghi được truy nhập tới theo tên trường.
- Các phần tử của bản ghi có thể nhận mọi kiểu của ngôn ngữ VHDL kể cả mảng và bảng ghi.

Ví dụ định nghĩa kiểu dữ liệu bảng ghi như sau:

```
type OPCODE is record
     PARITY   : bit;
     ADDRESS : std_logic_vector ( 0 to 3 );
     DATA_BYTE : std_logic_vector ( 7 downto 0 );
     NUM_VALUE : integer range 0 to 6;
     STOP_BITS : bit_vector (1 downto 0);
end record ;
. . .
signal  TX_PACKET, RX_PACKET  : OPCODE;
```



Cách truy nhập và gán dữ liệu cho các trường của bản ghi: Các phần tử của bản ghi được truy nhập theo tên bản ghi và tên trường, 2 thành phần này được ngăn cách bởi dấu '**.**'

```
TX_PACKET <= ( '1',"0011","11101010",5,"10" ) ;
TX_PACKET.ADDRESS <= ("0011");
TX_PACKET <= RX_PACKET;
```

```
TX_PACKET.ADDRESS <= RX_PACKET.ADDRESS;
```

c. Kiểu dữ liệu mảng 2 chiều (2-D Array)

Mảng 2 chiều là kiểu dữ liệu mảng của các phần tử mạng một chiều hay bảng ghi. Một số ví dụ định nghĩa và khai báo kiểu dữ liệu mảng 2 chiều như sau:

```
type Mem_Array is array (0 to 3) of std_logic_vector (7 downto 0);
type  Data_Array is array ( 0 to 2 ) of  OPCODE ;
...
signal My_Mem:Mem_Array ;
signal My_Data:Data_Array ;
```

Ví dụ ứng dụng dùng mảng 2 chiều khởi tạo một vùng nhớ ROM

```
constant My_ROM : REM_Array := (0 => (others=>'1'),
                                1 => "10100010",
                                2 => "00001111",
                                3 => "11110000");
```

d. Kiểu dữ liệu con:

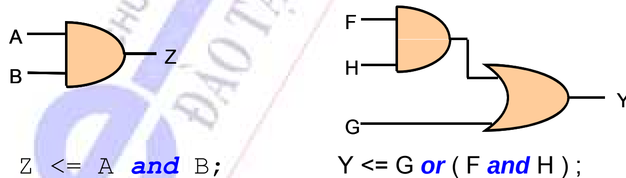
Là một tập hợp con của các kiểu dữ liệu đã được định nghĩa khác. Phép khai báo kiểu dữ liệu con có thể nằm ở mọi vị trí cho phép khai báo kiểu dữ liệu. Cú pháp khai báo chung:

Subtype Tên_kiểu_dữ_liệu_con **is** xác_định_kiểu_dữ_liệu_con;

```
Ví dụ:   subtype My_Int is integer range 0 to 255 ;
         subtype My_Small_Int is My_Int range 5 to 30 ;
         subtype word is bit_vector(31 downto 0);
```

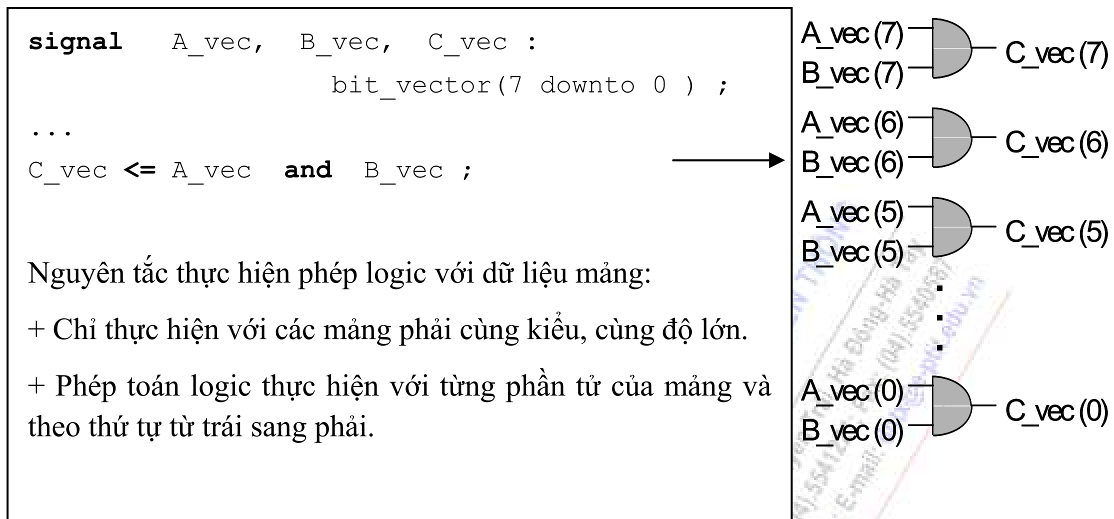
9.2.3 Các phép toán trong VHDL

Toán tử logic: được sử dụng cho các dạng dữ liệu là bit, boolean, bit_vector và std_logic_vector. Toán tử logic gồm có: **and, or, nand, nor, xor, not, xnor**.



Ví dụ: Z <= A *and* B;   Y <= G *or* (F *and* H) ;

- Toán tử logic dùng cho kiểu dữ liệu mảng:



Toán tử quan hệ: được sử dụng cho hầu hết các dạng dữ liệu, tất cả các toán tử quan hệ đều cho giá trị trả về dưới dạng boolean. Toán tử quan hệ gồm có: =, /=, <, <=, >, >=.

```
Ví dụ:     signal  FLAG_BIT : boolean ;
           signal  A, B : integer ;
           FLAG_BIT<=(A > B);
```

- Nguyên tắc thực hiện phép quan hệ với dữ liệu mảng:

+ Các mảng phải cùng kiểu, độ dài có thể khác nhau.

+ Mảng có độ dài khác nhau thì phép quan hệ thực hiện ưu tiên phần tử từ trái sang phải và so sánh theo giá trị ASCII.

```
signal   A_vec : bit_vector  ( 7 downto 0 ) := "11000110" ;
signal   B_vec : bit_vector  ( 5 downto 0 ) := "111001" ;
...
if ( A_vec  > B_vec )  then
       State <=  Normal;
else
       State <=  Code_Red;
end if;
```

Toán tử số học: được sử dụng cho số nguyên, số thực, và các dạng dữ liệu vật lý, std_logic. Std_logic_vector, Bit, Bit_vector. Cần chú ý rằng không phải tất cả toán tử số học đều có thể sử dụng cho mảng. Các toán tử số học là: +, -, *, /, abs (trị tuyệt đối), ** (hàm mũ).

Toán tử dịch: mỗi toán tử tác động vào thành phần bên trái của một toán hạng hoặc toán hạng bên phải của số nguyên để tạo ra rất nhiều toán tử dịch và quay. Số âm chỉ ra cách hướng khác được sử dụng. Mỗi toán tử cho kết qủa cùng dạng và kích thước với toán hạng ban đầu. Các toán tử dịch trong VHDL là: sll (dịch trái logic), srl (dịch phải logic), sla (dịch trái số học), sra (dịch phải số học), rol (quay trái), ror (quay phải).

```
Ví dụ: signal A_vec : bit_vector (7 downto 0) := "11000110";
```

```
signal D_vec : bit_vector (7 downto 0);
```

```
D_vec <= A_vec sll  2;
D_vec <= A_vec sra  2;
D_vec <= A_vec ror  3;
D_vec <= A_vec srl  2;
D_vec <= A_vec sra -2;
```



```
D_vec ="00011000"
D_vec ="11110001"
D_vec ="11011000"
D_vec ="00110001"
D_vec ="00011000"
```

Toán tử ghép nối: toán tử "&" cho phép ghép nối một cách linh hoạt các dữ liệu đơn và dữ liệu dạng mảng thành các mảng lớn hơn.

Ví dụ: signal A_vector, B_vector: std_logic_vector (7 downto 0);

signal Z_vector: std_logic_vector (15 downto 0);

Z_vector <= A_vector & B_vector;

Toán tử tách: cho phép ta lấy ra một số thành phần của mảng, chiều chỉ số của phép tách phải cùng chiều đánh chỉ số đã định nghĩa cho mảng.

```
Ví dụ: signal  Z_vec: std_logic_vector (15 downto 0);
       signal  B_vec: std_logic_vector (7 downto 0);
       B_vec <= Z_vec (12 downto 5);
```

Toán tử thuộc tính: Xác định thuộc tính dữ liệu của đối tượng biến và tín hiệu. Cú pháp chung:

Đối_tượng'thuộc_tính

- Các thuộc tính được định nghĩa trước cho kiểu dữ liệu mảng trong VHDL là:

+ **left, right**: trả lại chỉ số của phần tử bên trái nhất hoặc bên phải nhất của dữ liệu mảng.

+ **high, low** : trả lại chỉ số của phần tử cao nhất hoặc thấp nhất của kiểu dữ liệu mảng.

+ **range, reverse_range :** xác định khoảng của chỉ số của mảng.

+ **length :** trả về số lượng các phần tử của mảng.

+ **event, stable** : thuộc tính chỉ dùng cho đối tượng là tín hiệu, trả về giá trị boolean, chỉ ra rằng trên đường tín hiệu đang xét có xuất hiện sự kiện thay đổi hay giá trị trên đường tín hiệu ổn định tại thời điểm hiện tại. Các thuộc tính này dùng nhiều với lệnh **wait** và **if**. Ví dụ sử dụng toán tử thuộc tính như sau:

```
signal a : std_logic:='0';
...
PROCESS(a)
  TYPE bit4 IS ARRAY(0 TO 3) of BIT;
  TYPE bit_strange IS ARRAY(10 TO 20) OF BIT;
  VARIABLE len1, len2 : INTEGER;
BEGIN
  If (a'event and a='1')then -- sự kiện có xườn dương của a.
        len1 := bit4'LENGTH; -- returns 4
```

```
            len2 := bit_strange'LENGTH; -- returns 11
        End if;
    END PROCESS;
```

9.2.4 Các đơn vị thiết kế trong VHDL:

VDHL sử dụng 6 đơn vị thiết kế gồm 2 loại: đơn vị cơ bản và đơn vị thiết kế thứ cấp:

- Đơn vị thiết kế cơ bản:

- **Library**: Cho phép tạo thư viện trong VHDL
- **Package**: Tạo các gói giữ liệu trong Library, như các khai báo các đối tượng, khai báo chương trình con, hàm...
- **Entity**: (Thực thể) - cho phép khai báo các giao diện của một khối thiết kế số nào đó: như khai báo các chân vào/ra, các tham số của khối mạch...

- Đơn vị thiết kế thứ cấp (Phụ thuộc vào một đơn vị thiết kế cơ bản):

- **Architecture**: Mô tả hoạt động bên trong của một Entity hay đây chính là phần mô tả hoạt động của khối mạch số.
- **Package Body**: Mô tả chi tiết cho các khai báo trong Package như viết các hàm, các thủ tục ...
- **Configuration**: Đơn vị thiết kế cấu hình cho phép gắn các phiên bản của thực thể vào những kiến trúc khác nhau. Cấu hình cũng có thể được sử dụng để thay thế một cách nhanh chóng các phần tử của thực thể trong các biểu diễn cấu trúc của thiết kế.

+ ***Entity - (Thực thể) :***

Khai báo thực thể trong VHDL phần định nghĩa các chỉ tiêu phía ngoài của một phần tử hay một hệ thống. Thực chất của việc khai báo thực thể chính là khai báo giao diện của hệ thống với bên ngoài. Ta có thể có tất cả các thông tin để kết nối mạch vào mạch khác hoạt thiết kế tác nhân đầu vào phục vụ cho mục đích thử nghiệm. Tuy nhiên hoạt động thật sự của mạch không nằm ở phần khai báo này. Cú pháp khai báo chung của một Entity như sau:

```
entity Tên_th•c_th• is
      generic(--Khai báo danh sách các tham s• generic
         Tên_tham_s• : [Ki•u_d•_li•u] [:=giá_tr•_kh•i_t•o];
         ...
            );
      port(-- Khai báo danh sách ••i t••ng các port vào ra
          Tên_c•ng : [mode] [Ki•u_d•_li•u] [:=giá_tr•_kh•i_t•o];
          ...
         );
end Tên_th•c_th•;
```

Trong khai báo trên:

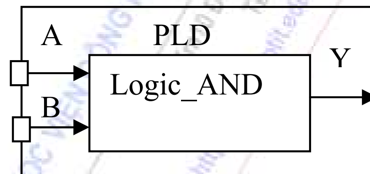
+ Tham số khai báo trong phần **generic** dùng để kiểm soát cấu trúc, hoạt động của thực thể, chúng sẽ được truyền giá trị hoặc lấy giá trị mặc định ban đầu khi thực thể được khởi tạo.

+ "--" Dấu đánh dấu dòng chú thích (comment) trong mã mô tả VHDL

+ [mode]: chỉ hướng tín hiệu của cổng có thể là: (**in, out, inout** hoặc **buffer**). Trong đó cổng dạng **in** chỉ dùng để đọc dữ liệu. Cổng dạng **out** chỉ dùng để gán giá trị dữ liệu. Cổng **inout** cho phép đồng thời vừa đọc vừa gán giá trị dữ liệu ở trong và ngoài chương trình. Cổng dạng **buffer** cho phép cả 2 thao tác đọc và gán dữ liệu từ bên trong chương trình, nhưng chỉ cho phép đọc dữ liệu từ ngoài chương trình.

```
Ví dụ khai báo thực thể cho một cổng logic AND:

entity Logic_AND is
      port(A, B : in std_logic ;
           Y    : out std_logic) ;
end Logic_AND;
```



+ ***Architecture – (Kiến trúc) :***

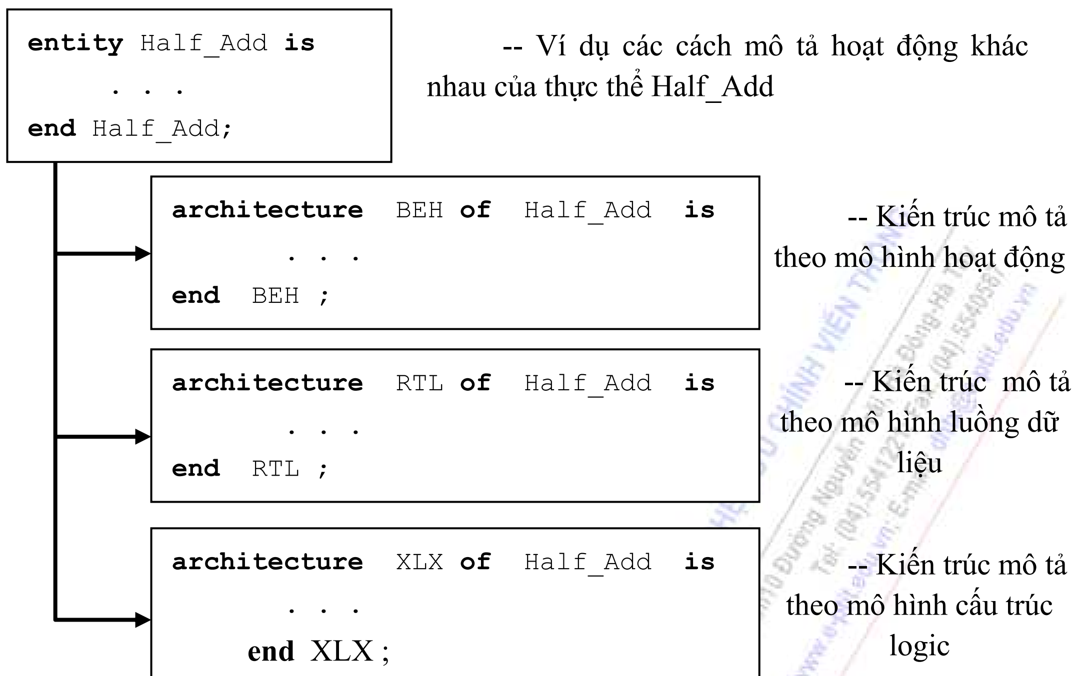
Cấu trúc này cho phép mô tả hoạt động bên trong của thực thể. Cú pháp chung của một **Architecture**:

```
Architecture Tên_kiến_trúc of Tên_thực_thể is
      -- Thực hiện các khai báo cho kiến trúc
      ...
Begin
      -- Viết các mô tả hoạt động bên trong cho thực thể
      ...
End Tên_kiến_trúc;
```

Phần khai báo kiến trúc có thể bao gồm các khai báo về các đối tượng signal, constant, kiểu dữ liệu, khai báo các phần tử bên trong hệ thống (component), hay các hàm (function) và thủ tục (proceduce) sử dụng để mô tả hoạt động của hệ thống. Tên của kiến trúc là nhãn được đặt tuỳ theo người sử dụng

VHDL cho phép tạo ra nhiều mô tả **Architecture** cho một thực thể, cho phép thực hiện nhiều cách mô tả hoạt khác nhau cho một thực thể. Mỗi cách mô tả hoạt động sẽ tối ưu về mặt thời gian thiết kế hay độ tin cậy hay tối ưu về tài nguyên sử dụng khi tổng hợp…

Có 3 cách chính mô tả kiến trúc của một phần tử (hoặc hệ thống số) đó là mô hình hoạt động (Behaviour), mô tả theo mô hình cấu trúc logic (Structure), và mô hình luồng dữ liệu. Tuy nhiên để mô tả cho một hệ thống, trong một kiến trúc có thể kết hợp sử dụng 2 hoặc cả 3 mô hình mô tả trên để thực hiện cho từng thành phần con tương ứng của hệ thống số. Trong phần sau của tài liệu này sẽ trình bày chi tiết hơn các phương pháp mô tả này.



+ ***Package*** và ***Package Body***

Package (gói dữ liệu) là đơn vị thiết kế cơ bản dùng để chứa những khai báo cho các đối tượng, khai báo chương trình con, hàm, kiểu dữ liệu, component có thể dùng chung cho những thiết kế, project, cấu trúc.

Package Body là đơn vị thiết kế phụ thuộc được dùng để chứa những mô tả chi tiết cho các khai báo trong đơn vị thiết kế **Package** nào đó, mô tả chi tiết nội dung của các hàm, các thủ tục ... **Package Body** thường được viết ngay sau **Package**. Cú pháp chung các đơn vị thiết kế **Package** và **Package Body** :

```
package My_Pack is

constant. . .
 . . .
function bv_to_integer (BV: bit_v..
                   return  integer
. . .
component . . .
. . .
subtype. . .

end package My_pack;

package body My_Pack is

       function bv_to_integer (BV: bit_v..
                   return  integer is
       variable ...
           begin
              for index in BV'range loop
               . . . .
            end function;
. . .
end  My_Pack ;
```

```
-- Cách sử dụng package trong
file mô tả VHDL.
library IEEE;-- Thư việc chuẩn
use IEEE.std_logic_1164.all ;
. . .
-- Trong phần mềm thiết kế ISE
gói dữ liệu do người sử dụng
tạo ra thường được tổ chức mặc
định trong thư viện "work"
use work.My_Pack.all;

entity . . .
```

+ Library (thư viện)

Trong VHDL có các thư viện thiết kế chuẩn, ngoài ra người thiết kế có thể tạo các thư viện thiết kế riêng. Trong một thiết kế VHDL nhiều đoạn chương trình có thể được gọi từ các thư viện khác nhau.

Phân tích VHDL là một quá trình kiểm tra các đơn vị thiết kế VHDL để cho đúng cú pháp và ngữ nghĩa, các đơn vị thiết kế VHDL được lưu vào thư viện để sử dụng sau này. Thư viện thiết kế chứa các những phần tử thư viện sau:

- **Package**: chứa những mô tả khai báo được dùng chung.
- **Entity**: là những mô tả giao diện thiết kế được dùng chung.
- **Architecture**: những mô tả hoạt động thiết kế được dùng chung.
- **Configuration**: là những phiên bản của thực thể được dùng chung.

Các đơn vị thư viện là các cấu trúc VHDL có thể được phân tích riêng rẽ theo trình tự nhất định.

Trong VHDL có thư viện thiết kế đặc biệt có tên là "WORK". Khi người thiết kế biên dịch một chương trình viết trên VHDL nhưng không chỉ rõ thư viện đích, chương trình này sẽ được biên dịch và chứa vào thư viện "WORK".

Ví dụ cách gọi và sử dụng thư viện như sau:

```
library  My_Lib ;
use  My_Lib.Fast_Counters.all ;

entity Mod1 is
   port ( . . .
```

+ ***Configuration (Cấu hình)***

Một thực thể có thể có một vài kiến trúc mô tả hoạt động cho nó. Trong quá trình thiết kế có thể phải thử nghiệm một vài biến thể của thiết kế bằng cách sử dụng các kiến trúc khác nhau. Cấu hình là thành phần cơ bản của đơn vị thiết kế. Cấu hình cho phép gắn các phiên bản của thực thể vào những kiến trúc khác nhau. Cấu hình cũng có thể được sử dụng để thay thế một cách nhanh chóng các phần tử của thực thể trong các biểu diễn cấu trúc của thiết kế.

Cú pháp của mô tả cấu hình như sau:

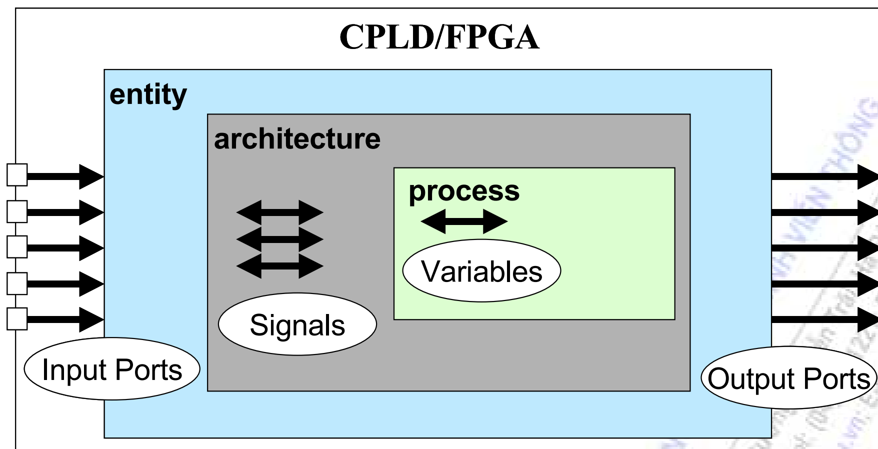
```
Configuration tên_cấu_hình of tên_thực_thể is
-- Phần khai báo của cấu hình (cho phép sử dụng
-- các phần tử trong package và library.
for đặc_tả_của_khối
      {mệnh_đề_use}
      {các_phần_tử_của_cấu_hình}
end for;
```

Ví dụ:

```
library ttl, work;
configuration v4_27_87 of processor is
      use work.all;
for structure_view
for a1:alu
use configuration ttl.sn74ls181;
end for;
for m1,m2,m3: mux
use entity multiplex4 (behavior);
end for;
for all: latch -- use defaults
end for;
end for;
end configuration v4_27_87;
```

9.2.5 Cấu trúc chung của một chương trình mô tả VHDL

Mô hình cấu trúc mô tả phần cứng số và phạm vi sử dụng của các đối tượng trong VHDL có thể được tổng kết đơn giản như trong hình 9-1 dưới đây:



Hình 9-1. Cấu trúc mô tả phần cứng và các đối tượng trong VHDL.

Sau đây là cấu trúc chung đơn giản của một chương trình mô tả VHDL:

```
-- Ví dụ cấu trúc 1 file mô tả cho một hệ thống phần cứng số dùng VHDL
-- Khai báo thư viện,(mặc định cần khai báo thư viện IEEE (thư viện
-- chuẩn đã được xây dựng).
library IEEE;...
-- Khai báo gói dữ liệu (package) trong thư viện cần sử dụng:
use IEEE.STD_LOGIC_1164.ALL;...
-- Khai báo thực thể
Entity Tên_thực_thể is
      -- Khai báo các tham số generic nếu cần:
      Generic( -- khai báo danh sách các tham số);
      Port(-- Khai báo danh sách các cổng vào/ra
          );
End Tên_thực_thể;
-- Bắt đầu viết
Architecture Tên_kiến_trúc of Tên_thực_thể is
{Khai báo:kiểu dữ liệu, các component,các đối tượng constant, signal}
Begin
      { Viết các mô tả dùng cấu trúc lệnh song song }
      ...
    Process(-- danh sách tín hiệu kích thích nếu cần)
      {Khai báo:kiểu dữ liệu, các đối tượng biến constant, variable }
    Begin
      { Viết các mô tả dùng cấu trúc lệnh tuần tự }
    End process;
```

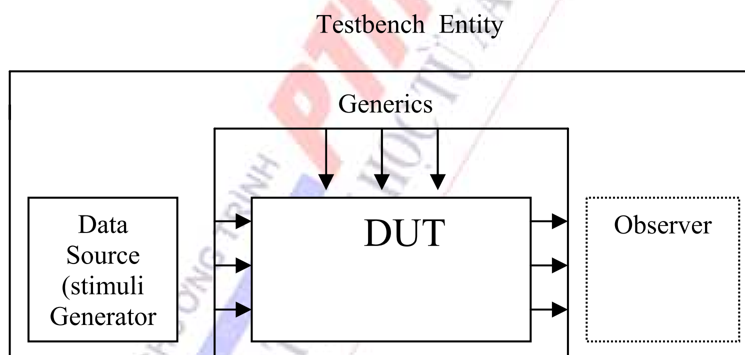
```
    ...
  { Viết các mô tả dùng cấu trúc lệnh song song hay process khác }
    ...
End Tên_kiến_trúc;
```

9.2.6 Môi trường kiểm tra testbench

Một trong các nhiệm vụ rất quan trọng là kiểm tra bản mô tả thiết kế. Kiểm tra một mô hình VHDL được thực hiện bằng cách quan sát hoạt động của nó trong khi mô phỏng và các giá trị thu được có thể đem so sánh với yêu cầu thiết kế.

Môi trường kiểm tra có thể hiểu như một mạch kiểm tra ảo. Môi trường kiểm tra sinh ra các tác động lên bản thiết kế và cho phép quan sát hoặc so sánh kết quả hoạt động của bản mô tả thiết kế. Thông thường thì các bản mô tả đều cung cấp chương trình thử. Nhưng ta cũng có thể tự xây dựng chương trình thử (testbench). Mạch thử thực chất là sự kết hợp của tổng hợp nhiều thành phần. Nó gồm ba thành phần. Mô hình VHDL đã qua kiểm tra, nguồn dữ liệu và bộ quan sát. Hoạt động của mô hình VHDL được kích thích bởi các nguồn dữ liệu và kiểm tra tính đúng đắn thông qua bộ quan sát. Hình 9-2 là sơ đồ tổng quát của một chương trình thử (Testbench).

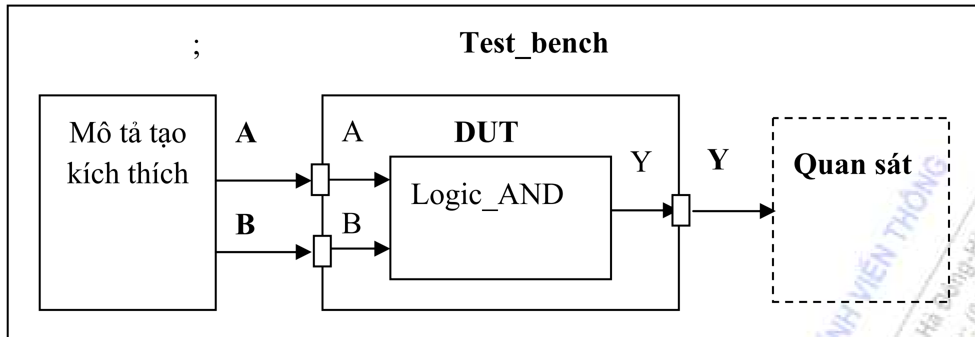
Testbench được mô tả như một Entity không có đầu vào đầu ra, chỉ có tín hiệu bên trong được ghép tới khối DUT cần được kiểm tra **theo kiểu cấu trúc**. Người thiết kế sẽ mô tả các tín hiệu bên trong này tạo ra tín hiệu kích thích cho các đầu vào của DUT và đọc kết quả ra để quan sát...



Trong đó: DUT: (device under test) mô hình VHDL cần kiểm tra
Observer: khối quan sát kết quả
Data source: nguồn dữ liệu (khối tạo ra các tín hiệu kích thích)

Hình 9.2. Sơ đồ tổng quát chương trình thử Testbench

Viết Testbench cho thực thể Logic_AND đã mô tả ở phần trước:



Thực thể **Test_bench** không có các cổng vào ra mà chỉ khai báo các tín hiệu nội bộ A, B, Y để nối tới khối DUT cần kiểm tra (Logic_AND). Trong phần kiến trúc mô tả hoạt động của Test_bench, coi khối Logic_AND như một component để tạo thành khối Test_bench. Toàn bộ mã mô tả cho Test_bench như sau:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
-- Khai báo thực thể Test_bench;
ENTITY Test_bench IS
END Test_bench;
-- Mô tả kiến trúc của Test_bench
ARCHITECTURE behavior OF Test_bench IS
      COMPONENT Logic_AND
      PORT(
            A : IN std_logic;
            B : IN std_logic;
            Y : OUT std_logic
            );
      END COMPONENT;
      SIGNAL A :  std_logic:='0';
      SIGNAL B :  std_logic:='0';
      SIGNAL Y :  std_logic;
BEGIN
-- Nối chân cổng vào ra của DUT với các tín hiệu của Test_bench
   uut: Logic_AND PORT MAP(
            a => a,
            b => b,
            y => y
      );
   tb : PROCESS
   BEGIN
       -- Viết mô tả tạo kích thích
```

```
            A<= '1' after 10ns;
            B<= '1' after 20ns;
                                         ...
   END PROCESS;
-- *** End Test Bench - User Defined Section ***
END;
```

Trong các phần mềm thiết kế sau khi hoàn thành các mô tả cho Test_bench, người thiết kế sẽ chạy công cụ mô phỏng, các tín hiệu đầu ra của DUT sẽ được mặc tính đọc ra và cho phép người thiết kế quan sát dễ dàng dưới dạng giản đồ thời gian, hay các file số liệu…

Người thiết kế có thể dễ dàng viết các mô tả kích thích để tạo ra các yêu cầu kiểm tra tùy ý cho bản thiết kế của mình. Nhiều chức năng mô phỏng, kiểm tra được hỗ trợ rất mạnh bởi các phần mềm thiết kế.

9.2.7 Các cấu trúc lệnh song song

Như đã trình bày trong phần cấu trúc chung của chương trình mô tả VHDL, trong mô tả một kiến trúc (Architecture) có chứa nhiều cấu trúc lệnh song song. Mỗi cấu trúc lệnh song song sẽ tương ứng với một thành phần phần cứng nào đó khi thực hiện tổng hợp mạch, mỗi cấu trúc song song có thể viết ở bất kỳ vị trí nào trong đoạn mô tả Architecture mà chức năng hoạt động của thực thể không thay đổi. Các cấu trúc lệnh song song có trong VHDL gồm:

+ Cấu trúc process.

+ Lệnh gán tín hiệu song song.

+ Lệnh gán có điều kiện.

+ Lệnh gán tín hiệu có lựa chọn.

+ Khối.

+ Phép gọi chương trình con song song.

a. Cấu trúc Process:

Cấu trúc Process được tạo thành từ một tập hợp cấu trúc lệnh tuần tự (được trình bày chi tiết ở phần sau). Nó là khối cơ bản của việc mô tả hoạt động của thực thể. Tất cả các

```
    [Nhãn] Process [(Danh sách tín hiệu kích thích)]
[ Khai báo:kiểu dữ liệu, các đối tượng biến constant, variable ]
    Begin
      { Viết các mô tả dùng cấu trúc lệnh tuần tự }
    End process;
```

Process trong một thiết kế được thực hiện song song. Tuy nhiên, tại một thời điểm xác định chỉ có một câu lệnh tuần tự được thực hiện trong mỗi cấu trúc Process. Cấu trúc tổng quát:

Trong đó các phần đặt trong dấu [ ] có thể có hoặc không.

- Nhãn_lệnh: Tuỳ thuộc người thiết kế đặt tên.

- Danh sách tín hiệu kích thích: Danh sách các yếu tố kích thích hoạt động

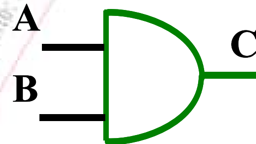
Nếu Process chứa (danh sách tín hiệu kích thích) thì lúc đó Process sẽ được thực hiện khi có bất kỳ sự thay đổi nào của bất kỳ tín hiệu nào trong danh sách tín hiệu kích thích. Điều này tương đương với Process không chứa danh sách tín hiệu kích thích nhưng lại chứa lệnh **wait** ở vị trí câu lệnh cuối cùng trong quá trình:

```
Wait on <danh sách tín hiệu kích thích>
```

Khi tổng hợp mạch thì mỗi Process sẽ tương ứng với một khối mạch chức năng nào đó. Còn khi thực hiện mô phỏng, việc thực hiện một Process bao gồm việc thực hiện lặp lại các cấu trúc lệnh tuần tự chứa bên trong thân của Process. Giống như một vòng lặp vô hạn và mỗi bước lặp được thực hiện mỗi khi có sự thay đổi của bất kỳ tín hiệu nào trong danh Sách tín hiệu kích thích.

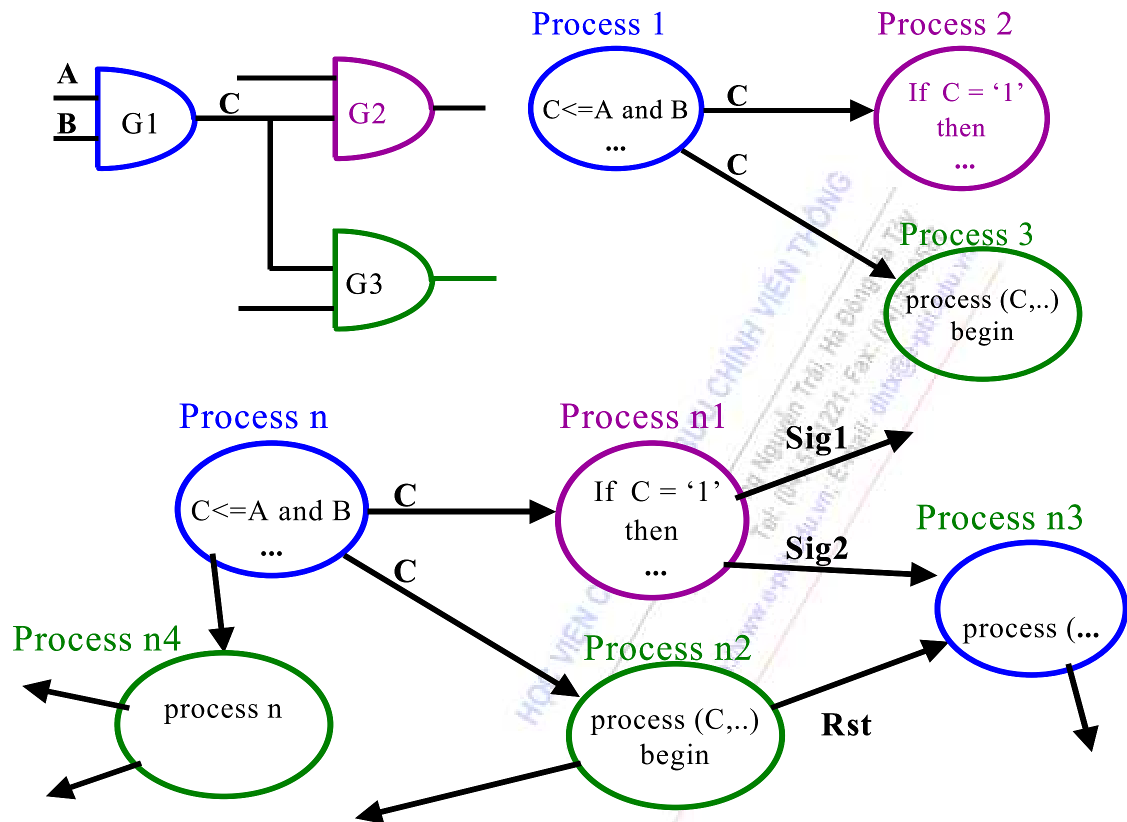
Ví dụ Process mô tả mạch logic AND như sau:

```
entity Logic_AND is
    Port ( A,B : in std_logic;
           C   : out std_logic);
end Logic_AND;
architecture Behavioral of Logic_AND is
begin
    Process(A,B)
    begin
           C<= A and B;
    end Process;
end Behavioral;
```



Một Process liên kết với phần còn lại của thiết kế thông qua các thao tác đọc các giá trị từ các tín hiệu vào, các cổng được khai báo ngoài Process và ghi giá trị vào các tín hiệu ra, cổng ra. Chú ý khi thiết kế là một tín hiệu có thể vào (được đọc ra) bởi nhiều Process nhưng chỉ nên được ghi ra bởi một Process.

Sự hoạt động đồng thời của mỗi Process và mô hình kết nối của chúng được mô tả như hình vẽ 9-3, trong đó tín hiệu sẽ truyền giá trị giữa những Process hoạt động đồng thời.:



Hình 9-3. Mô hình kết nối của các Process.

b. Các phép gán tín hiệu song song

Phép gán tín hiệu song song sử dụng bên trong các Architecture nhưng bên ngoài Process. Dạng đơn giản nhất của phép gán tín hiệu song song có cú pháp như sau:

<tín_hiệu_đích> <= <biểu_thức> [**after** <biểu_thức_thời_gian>];

Trong đó <tín_hiệu_đích> nhận giá trị của <biểu_thức>, chú ý là lệnh **after** chỉ dùng cho mô phỏng còn khi tổng hợp mạch nó sẽ được bỏ qua. Phép gán song song tương đương một Process chứa 1 phép gán tín hiệu.

Ví dụ mô tả mạch AND và OR có cùng 4 đầu vào như sau:

```
...
architecture Behavioral of logic1 is
  signal I1, I2, I3, I4, AND_out, OR_out: std_logic;
begin
    ...
    AND_out<= I1 and I2 and I3 and I4;
    OR_out<= I1 or I2 or I3 or I4;
    ...
end Behavioral;
```

Đoạn chương trình trên tương đương với đoạn chương trình VHDL với các Process chứa các phép gán tín hiệu tuần tự sau:

```
...
architecture Behavioral of logic1 is
  signal I1, I2, I3, I4, AND_out, OR_out: std_logic;
begin
    ...
    process(I1, I2, I3, I4)
    begin
            AND_out<= I1 and I2 and I3 and I4;
    end process;
    process(I1, I2, I3, I4)
    begin
            OR_out<= I1 or I2 or I3 or I4;
    end process;
    ...
end Behavioral;
```

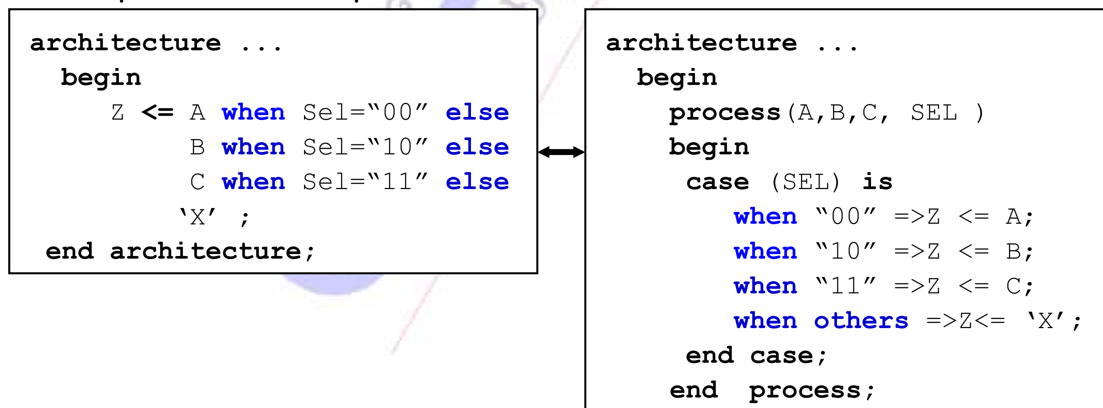

c. Phép gán tín hiệu có điều kiện

Phép gán tín hiệu có điều kiện là cấu trúc lệch song song thực hiện phép gán giá trị của các biểu thức cho một tín hiệu đích tùy theo các điều kiện đặt ra. Cú pháp chung:

```
<tín_hiệu_đích> <= <biểu_thức>[after <biểu_thức_thời_gian>] when <điều_kiện> else
                   <biểu_thức>[after <biểu_thức_thời_gian>] when <điều_kiện> else
                   ...
                   <biểu_thức>[after <biểu_thức_thời_gian>];
```

Cấu trúc phép gán tín hiệu có điều kiện có thể coi là cấu trúc song song của lệnh tuần tự **If** được thay thế tương đương với **Process** chứa lệnh tuần tự **if**.

Ví dụ mô tả cấu trúc chọn kênh như sau:



```
architecture ...
  begin
     Z <= A when Sel="00" else
          B when Sel="10" else
          C when Sel="11" else
         'X' ;
 end architecture;
```

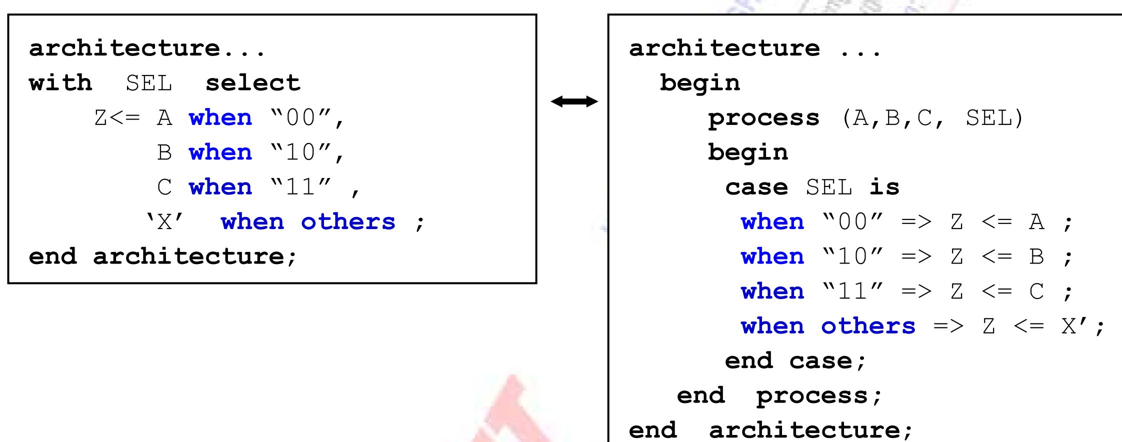
```
architecture ...
  begin
    process(A,B,C, SEL )
    begin
     case (SEL) is
        when "00" =>Z <= A;
        when "10" =>Z <= B;
        when "11" =>Z <= C;
        when others =>Z<= 'X';
     end case;
    end  process;
```

d. Phép gán tín hiệu theo lựa chọn

Phép gán tín hiệu theo lựa chọn thực hiện gán cho một tín hiệu đích với biểu thức **with.** Cấu trúc này có thể coi như là cấu trúc song song của lệnh tuần tự **case,** nó có thể thay thế tương đương với Process chứa lệnh tuần tự **case**. Cú pháp chung của lệnh **with** như sau:

```
With <biểu_thức_lựa_chọn> select
      <tín_hiệu_đích> <= <biểu_thức> [after <biểu_thức_thời_gian>]
                                          when <giá_trị_lựa _chọn>,
                         <biểu_thức> [after <biểu_thức_thời_gian>]
                                          when <giá_trị_lựa _chọn>,
                              ...
                         <biểu_thức> [after <biểu_thức_thời_gian>]
                                          when others;
```

Ví dụ mô tả cấu trúc chọn kênh như sau:



```
architecture...
with  SEL  select
    Z<= A when "00",
        B when "10",
        C when "11" ,
       'X'  when others ;
end architecture;
```

↔

```
architecture ...
  begin
     process (A,B,C, SEL)
     begin
      case SEL is
       when "00" => Z <= A ;
       when "10" => Z <= B ;
       when "11" => Z <= C ;
       when others => Z <= X';
      end case;
    end  process;
end  architecture;
```

e. Khối (Block)

Block bao gồm tập hợp các cấu trúc lệnh song song. Một kiến trúc có thể phân tách thành một số cá cấu trúc logic. Mỗi khối biểu diễn một thành phần của mô hình và thường được sử dụng để tổ chức một tập hợp các cấu trúc song song phân cấp. Cú pháp chung:

```
<nhãn>: Block
         {<phần_khai_báo>}
begin
         {<câu_lệnh_song_song>} - có trình tự bất kỳ
end block;
```

<phần_khai_báo> : xác định các đối tượng tồn tại cục bộ trong khối và có thể là các khai báo sau:

- Khai báo hằng, kiểu dữ liệu, tín hiệu.
- Thân chương trình con.
- Khai báo bí danh.
- Khai báo component.
- Luật use.

f. Gọi chương trình con song song

Phép gọi chương trình con song song tương đương với các process bao gồm các phép gọi chương trình con tuần tự tương ứng. Mỗi phép gọi chương trình con tương đương với một process không chứa dãy danh sách các tín hiệu kích thích, phần khai báo rỗng và phần thân chứa một phép gọi chương trình con, tiếp theo là một câu lệnh **wait**.

9.2.8 Cấu trúc lệnh tuần tự

Trong ngôn ngữ VHDL một cấu trúc đồng thời quan trọng là **Process**. Cấu trúc này được sử dụng để mô tả hành vi hoạt động của mạch số. Trong kiến trúc, tất cả các Process sẽ được tổng hợp thành một khối mạch chức năng và thực hiện đồng thời khi mô phỏng. Một Process n được xây dựng từ các cấu trúc lệnh tuần tự. Khi mô phỏng các lệnh tuần tự được thực hiện lần lượt trong một chu trình vô hạn bắt đầu từ lệnh thứ nhất đến lệnh cuối và được kích hoạt trở lại thực hiện lệnh đầu mỗi khi có bất kỳ sự thay đổi nào trong danh sách tín hiệu kích thích hay trong danh sách tín hiệu trong câu lệnh **wait**.

Các cấu trúc lệnh tuần tự cơ bản trong VHDL gồm:

- Câu lệnh gán cho biến.
- Câu lệnh gán cho tín hiệu.
- Câu lệnh **if**.
- Câu lệnh **case**.
- Câu lệnh rỗng **Null**.
- Các lệnh lặp.

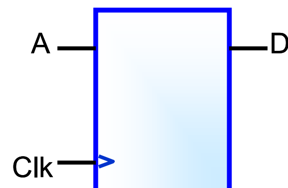
a. Phép gán biến

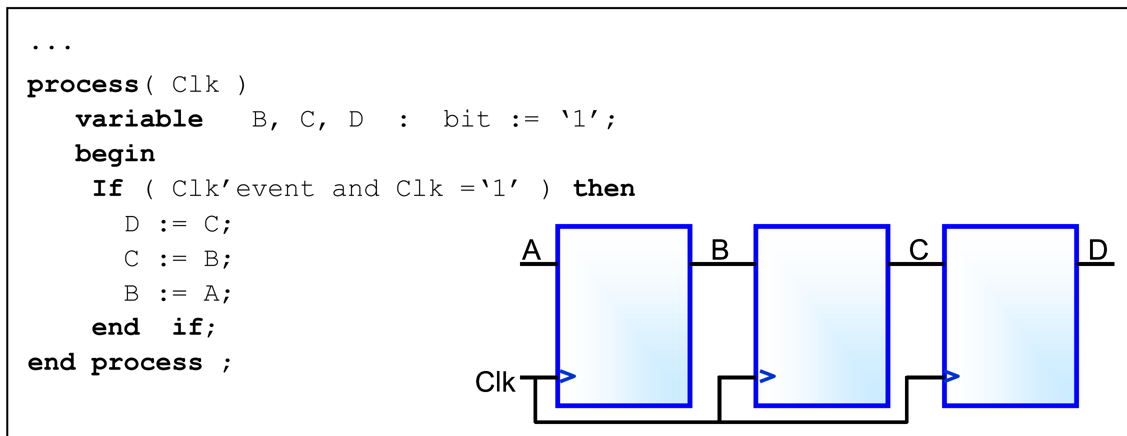
Cú pháp của phép gán biến như sau:

biến := biểu_thức

Phép gán biến được thực hiện với thời gian mô phỏng bằng 0, và giá trị biến sẽ được cập nhật ngay giá trị của biểu thức. Đối tượng <biến> chỉ được khai báo và sử dụng trong Process và chương trình con, nó được sử dụng để lưu trữ các kết quả trung gian. Ví dụ:

```
...
process( Clk )
  variable   B, C, D  : bit := '1' ;
  begin
   If  (Clk'event and Clk ='1')  then
     B := A ;
     C := B ;
     D := C ;
   end  if  ;
end  process ;...
```





```
...
process( Clk )
   variable   B, C, D  :  bit := '1';
   begin
    If ( Clk'event and Clk ='1' ) then
      D := C;
      C := B;
      B := A;
    end  if;
end process ;
```


Chú ý trong 2 ví dụ trên, giá trị các biến được cập nhập tức thì, ví dụ thứ nhất khi tổng hợp mạch chỉ tạo ra một triger D. Còn với ví dụ thứ 2 thứ tự gán biến thay đổi, kết quả tạo ra 3 triger D. Trong ví dụ 1, nếu B,C,D là tín hiệu thì kết quả hoàn toàn khác. Xem ví dụ ở phần phép gán tín hiệu.

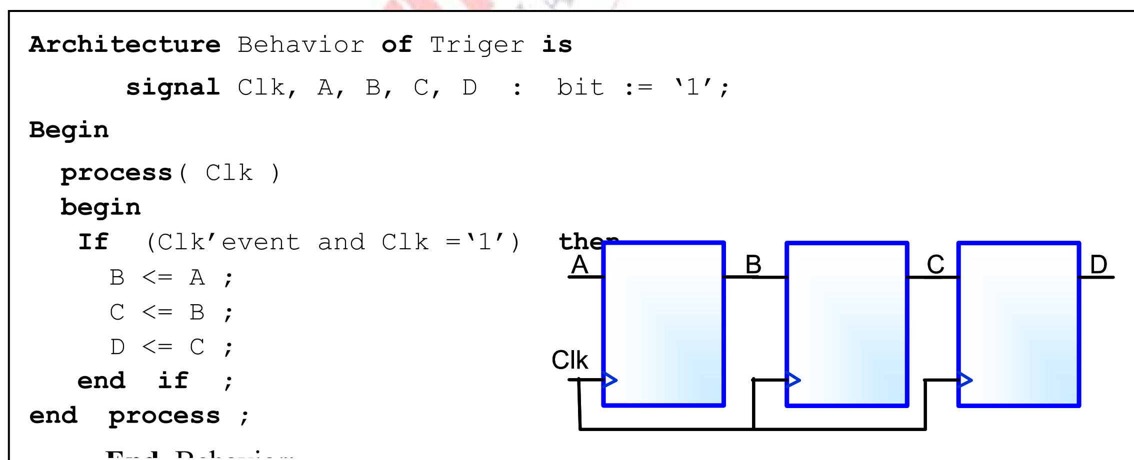
b. Phép gán tín hiệu

Cú pháp của phép gán tín hiệu như sau:

Tín_hiệu_đích<= biểu_thức [**after** giá_trị_thời_gian];

Khác với phép gán biến, phép gán tín hiệu trong **Process** không được cập nhập ngay tức thì mà phép gán đó chỉ được đặt kế hoạch thực hiện và kết quả chỉ được cập nhập sau khi kết thúc **Process**.

Ví dụ:



```
Architecture Behavior of Triger is
      signal Clk, A, B, C, D  :  bit := '1';
Begin
  process( Clk )
  begin
   If   (Clk'event and Clk ='1')   then
     B <= A ;
     C <= B ;
     D <= C ;
   end  if  ;
end  process ;
```


c. Lệnh if

Lệnh này cho phép các phép toán được thực hiện trên một điều kiện nào đó. Có ba dạng cơ bản là:

```
+ Dạng 1:
   if (Điều_kiện) then
         <Các_câu_lệnh_tuần_tự>;
   end if;
+ Dạng 2:
```

```
        if (Điều_kiện) then
              <Các_câu_lệnh_tuần_tự>;
        else
              <Các_câu_lệnh_tuần_tự>;
        end if;
    + Dạng 3:
        if (Điều_kiện_1) then
              <Các_câu_lệnh_tuần_tự>;
        elsif  (Điều_kiện_2) then
              <Các_câu_lệnh_tuần_tự>;
        elsif  (Điều_kiện_3) then
              <Các_câu_lệnh_tuần_tự>;
        else
              <Các_câu_lệnh_tuần_tự>;
        end if;
```

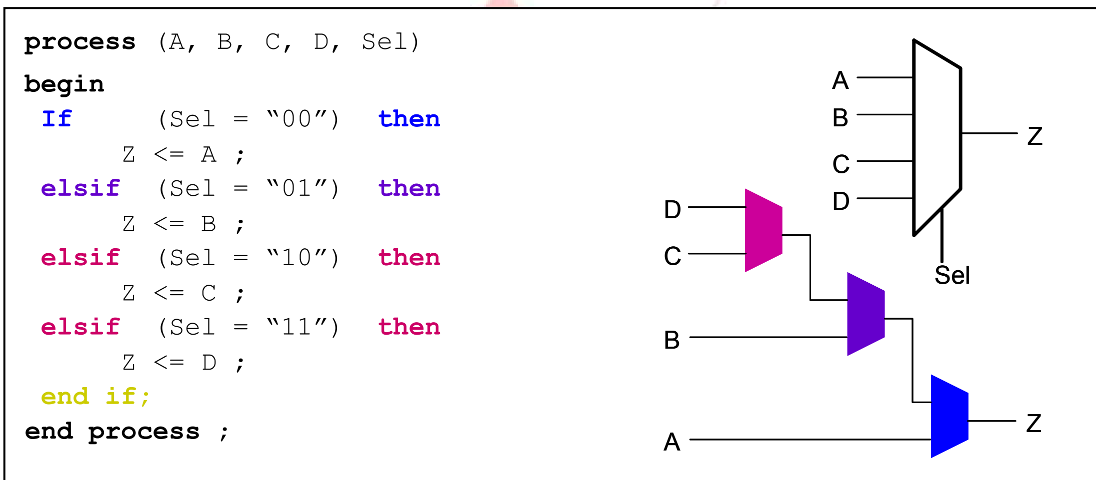
Trong lệnh **if/else**, ta phải chú ý một số điều sau:

+) Điều kiện đúng đầu tiên được tìm thấy sẽ được thực hiện.

+) Các điều kiện có thể chồng lấp lên nhau.

+) Điều kiện đầu tiên trong lệnh **if/else** được ưu tiên.

Ví dụ:



d. Lệnh case:

Lệnh **case** được sử dụng trong trường hợp có một biểu thức để kiểm soát nhiều rẽ nhánh trong chương trình VHDL. Các lệnh tương ứng với một trong các lựa chọn sẽ được thực hiệu nếu biểu thức kiểm soát có giá trị bằng giá trị tương ứng của lựa chọn đó. Có hai dạng cơ bản:

```
Dạng 1:
      Case (biểu_thức_kiểm_soát) is
            When <giá_trị_lựa_chọn> =>
                  <Các_câu_lệnh_tuần_tự>;
            When <giá_trị_lựa_chọn> =>
```

```
                <Các_câu_lệnh_tuần_tự>;
          ...
      end case;
Dạng 2:
      Case (selector expression) is
            When <giá_trị_lựa_chọn> =>
                  <Các_câu_lệnh_tuần_tự>;
            When <giá_trị_lựa_chọn> =>
                  <Các_câu_lệnh_tuần_tự>;
            ...
            When others =>
                  <Các_câu_lệnh_tuần_tự>;
            end case;
```

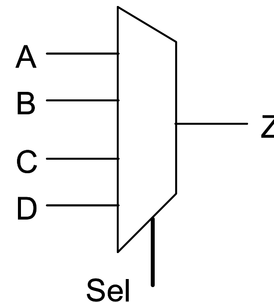
Các chú ý khi dùng lệnh case:

+) Tất cả các giá trị của biểu thức lựa chọn phải được chỉ rõ.

+) Không có các giá trị lựa chọn bị chồng lấp lên nhau.

Ví dụ:

```
process   (A, B, C, D, Sel )
begin
case   Sel   is
   when   "00"   =>   Z   <= A ;
   when   "01"   =>   Z   <= B ;
   when   "10"   =>   Z   <= C ;
   when   "11"   =>   Z   <= D ;
end   case ;
end process ;
```



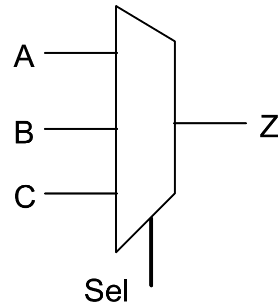
e. Câu lệnh rỗng Null

Câu lệnh rỗng có cú pháp như sau:

Null;

Trong VDHL khi chương trình mô phỏng gặp câu lệnh **Null** nó sẽ bỏ qua lệnh này và thực hiện lệnh tiếp theo sau. Thông thường lệnh **Null** dùng để chỉ trường hợp không thực hiện của lệnh một cách tường minh khi có các điều kiện trả lại giá trị true. Do đó lệnh Null thường được dùng trong các câu lệnh **case** đối với những giá trị lựa chọn không cần thao tác. Ví dụ:

```
process  (A, B, C, D, Sel )
begin
case  Sel  is
   when  "00"  =>   Z  <= A ;
   when  "01"  =>   Z  <= B ;
   when  "10"  =>   Z  <= C ;
   when  others  =>  Null;
end  case ;
end process ;
```



f. Các lệnh lặp

Lệnh lặp **loop** chứa thân vòng lặp bao gồm dãy các câu lệnh sẽ được thực hiện không hoặc nhiều lần. Cú pháp của lệnh lặp như sau:

```
[<nhãn>:] [<sơ_đồ_lặp>] loop
      {<lệnh_tuần_tự>}|
      {next [<nhãn>] [when <điều_kiện>];}|
      {exit [<nhãn>] [when <điều_kiện>];}
end loop [nhãn];
```

- <nhãn>: nhãn của vòng lặp và thường được dùng để xây dựng những vòng lặp lồng nhau, trong đó mỗi vòng lặp được kết thúc bởi từ khóa **end loop**.

- <sơ_đồ_lặp>: vòng lặp với sơ đồ lặp **for** hoặc vòng lặp **while**, và vòng lặp không chứa các sơ đồ lặp.

Với những vòng lặp không chứa [<sơ_đồ_lặp>], các lệnh trong dãy lệnh tuần tự sẽ được thực hiện cho tới khi được ngắt bởi câu lệnh **exit**. Trong đó câu lệnh **next** cũng được dùng để thay đổi trình tự thực hiện thân của vòng lặp.

Ví dụ vòng lặp không chứa sơ đồ lặp:

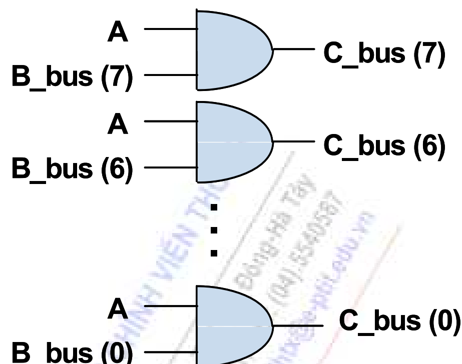
```
Count_down: Process
      Variable Min,Sec: integer range 0 to 60;
Begin
      L1: loop
            L2: loop
                  exit L2 when (Sec=0);
                  wait until CLK'event and CLK='1';
                  Sec:=Sec-1;
            End loop L2;
            Exit L1 when (Min=0);
            Min:=Min-1;
            Sec:=60;
      End loop L1;
End process Count_down;
```

Ví dụ vòng lặp chứa <sơ_đồ_lặp> dạng **for**:

```
process   ( A, B_bus )
begin
 for   i in 7 downto 0   loop
    C_bus (i) <= A  and  B_bus (i);
 end  loop ;
end process;

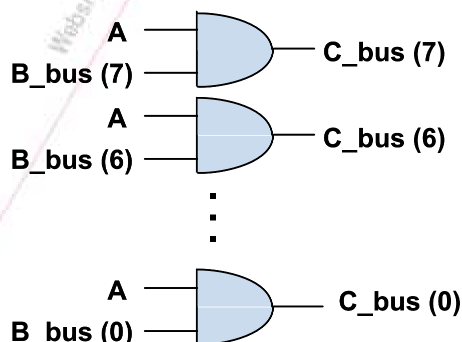
hoặc:

process   ( A, B_bus )
begin
 for   i in 0 to 7   loop
    C_bus (i) <= A  and  B_bus (i);
 end  loop ;
end process;
```



Ví dụ vòng lặp chứa <sơ_đồ_lặp> dạng **while** như sau:

```
process   ( A, B_bus )
      variable i:integer:=0;
begin
 while (i<8) loop
    C_bus (i) <= A  and  B_bus (i);
    i:=i+1;
 end  loop ;
end process;
```



9.3. CÁC MỨC ĐỘ TRỪU TƯỢNG VÀ PHƯƠNG PHÁP MÔ TẢ HỆ THỐNG PHẦN CỨNG SỐ

Sử dụng VHDL cho phép mô tả hệ thống phần cứng số theo các mức độ trừu tượng khác nhau. Hình vẽ 9-4 mô tả các mức độ mô tả trừu tượng giảm dần khi sử dụng VHDL.

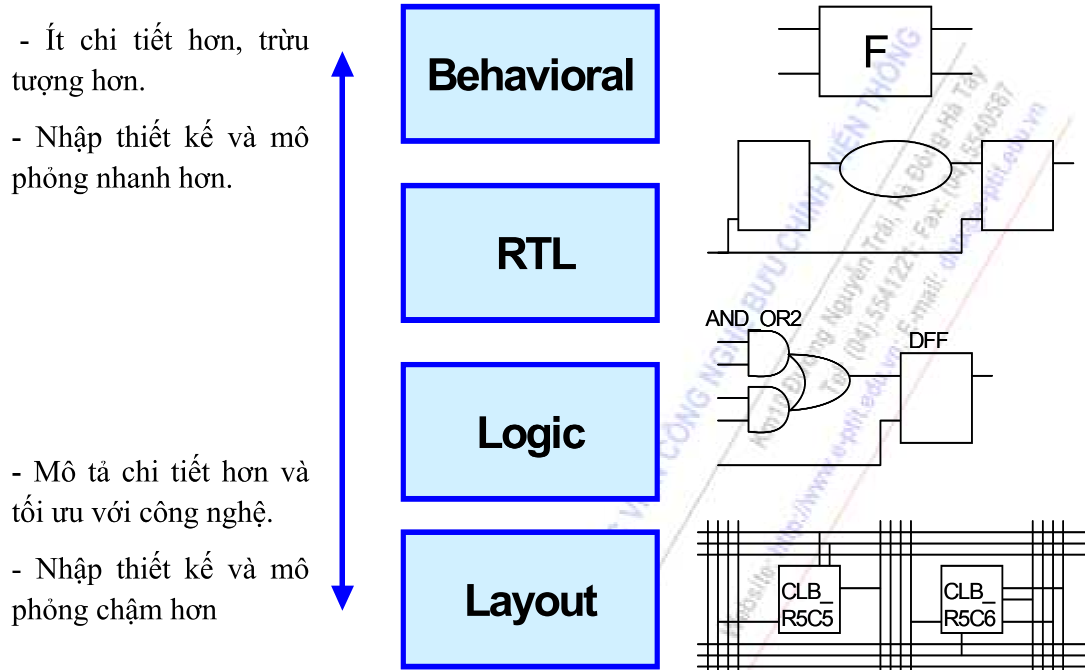
+ Mức mô tả theo mô hình hành vi (**Behavioral**): mức độ mô tả trừu tượng cao nhất, kiểu mô tả này thường dùng cho mô hình phần cứng và mô phỏng.

+ Mức mô tả theo mô hình luồng dữ liệu **RTL** (Register Tranfer Level): Kiểu mô tả này khá tối ưu và có cho khả năng tổng hợp cao, độc lập với công nghệ.

+ Mức mô tả theo mô hình cấu trúc **logic**: Kiểu mô tả này thường sử dụng các cấu trúc logic đã được xây dựng sẵn, hoặc chọn trong thư viện của nhà cung cấp phù hợp với loại công nghệ sử dụng.

+ Mức mô tả theo cấu trúc **layout**. Mức độ mô tả chi tiết nhất, mô tả chi tiết tới cấu trúc bên trong những tài nguyên đã sẵn có trong cấu kiện, cách này tối ưu cho việc tổng hợp trên loại cấu kiện, công nghệ đã sử dụng.

Với hệ thống số thông thường được mô tả theo 3 mức: hành vi, RTL và cấu trúc logic. Trong một thiết kế có thể chỉ sử dụng theo một cách mô tả, hoặc cũng có thể phải dùng kết hợp cả 3 cách tùy theo độ phức tạp của thiết kế, yêu cầu về thời gian thiết kế, yêu cầu về sự tối ưu phần cứng…



Hình 9-4. Các mức độ mô tả hệ thống phần cứng số.

9.3.1. Phương pháp mô tả theo mô hình cấu trúc logic

Mô hình cấu trúc của một phần tử (hoặc hệ thống) có thể bao gồm nhiều cấp cấu trúc bắt đầu từ một cổng logic đơn giản đến xây dựng mô tả cho một hệ thống hoàn thiện. Thực chất của việc mô tả theo mô hình cấu trúc là mô tả các phần tử con bên trong hệ thống và sự kết nối của các phần tử con đó. Cách thức mô tả cấu trúc của thành phần con cũng tương tự như cách thức mô tả thực thể. Trước hết để mô tả cấu trúc của thành phần con, chúng ta phải xác định rõ các giao diện của thành phần con. Các giao diện này chính là các đường tín hiệu vào và ra từ thành phần con.

Trước khi được sử dụng trong kiến trúc của cả hệ thống, các thành phần phải được khai báo một cách tường minh theo cú pháp sau:

```
Component <tên_thành_phần>
      Port(<khai_báo_danh_sách_các_cổng_cục_bộ;>)
      -- Tương tự như khai báo trong thực thể
End component;
```

Chú ý: Các cổng vào ra của mỗi thành phần con không được kết nối trực tiếp với nhau mà phải kết nối thông qua tín hiệu nội bộ có cùng kiểu, cùng độ lớn với các cổng vào ra đó.

Cú pháp mô tả móc nối giữa các thành phần con như sau:

```
<nhãn_khởi_tạo>:<tên_thành_phần>
```

```
port map ([<tên_cổng_cục_bộ> =>] <biểu_thức>
{[<tên_cổng_cục_bộ>=>]<biểu_thức>});
```

Cấu trúc **port map** ánh xạ các cổng của phần tử vào các tín hiệu. Ánh xạ này có thể hiểu như việc kết nối cổng tương ứng của phần tử vào đường tín hiệu. Cấu trúc port map đặt tương ứng mỗi cổng thực của phiên bản với một cổng cục bộ thành phần. Thực hiện nối các cổng vào ra của các thành phần con với các chân vào ra của hệ thống hoặc nối với tín hiệu nội bộ trong hệ thống để kết nối tới các cổng vào ra của các thành phần con khác. Ánh xạ được thực hiện theo vị trí theo tên:

+ Khi sử dụng ánh xạ theo vị trí, chúng ta đưa ra danh sách các tín hiệu tuân theo đúng trật tự mà cổng được khai báo.

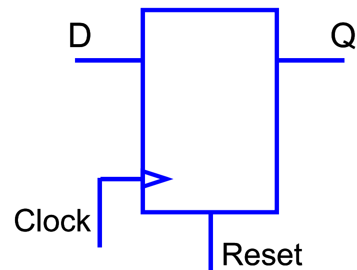
+ Đối với trường hợp ánh xạ theo tên, chúng ta sử dụng cấu trúc ánh xạ tường minh đặt tương ứng với mỗi cổng với các tín hiệu thực:

<tên_cổng_cục_bộ> => <tên_tín_hiệu_thực>

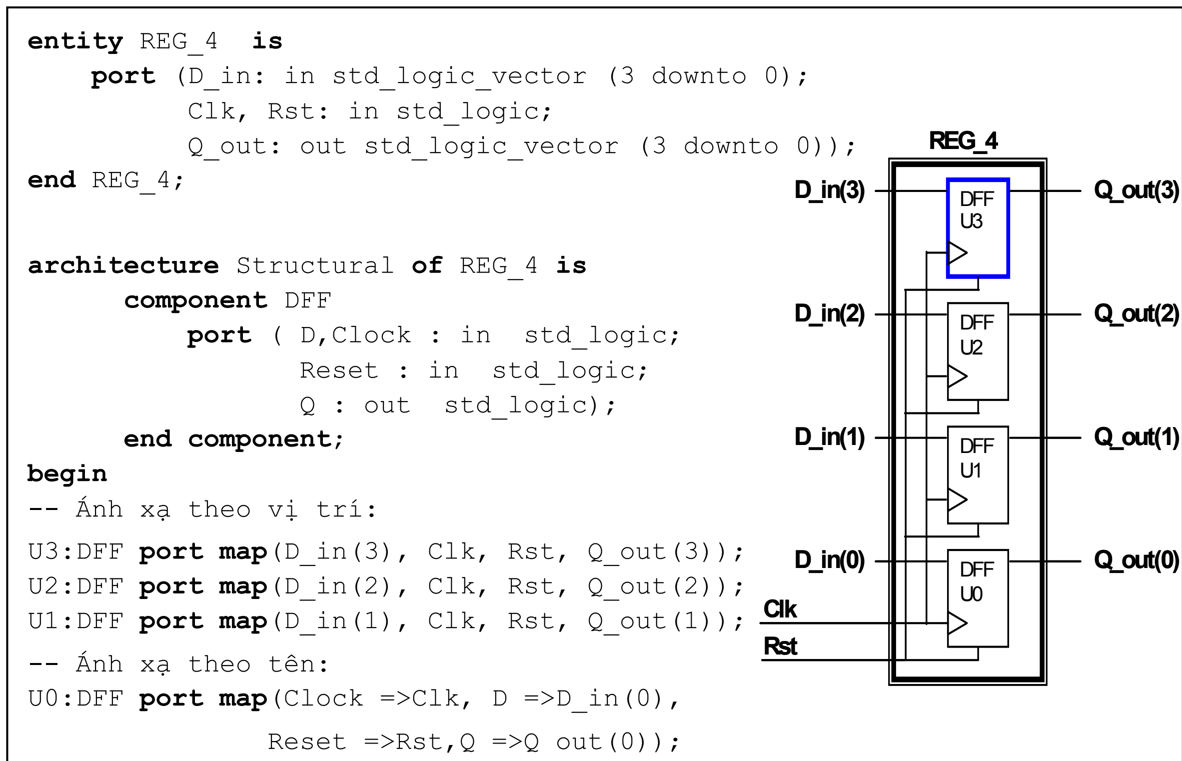
Ví dụ mô tả mô hình cấu trúc một thanh ghi 4 bít được xây dựng từ 4 triger D. Có thể mô tả triger D sau đó sau đó mô tả sơ đồ móc nối các phần tử triger D tạo thành thanh ghi.

- Mô tả triger D như sau:

```
entity DFF is
    port (  D, Clock : in std_logic ;
            Reset : in std_logic ;
            Q : out std_logic) ;
end entity DFF ;
architecture RTL of DFF is
begin
    process (Clock, Reset)
    begin
      If  (Reset = '1' ) then
       Q <= '0' ;
      elsif (Clock'event and Clock = '1') then
      Q <= D ;
      end if;
   end process ;
end architecture RTL;
```



- Ví dụ Mô tả cấu trúc của thanh ghi:



```
entity REG_4  is
    port (D_in: in std_logic_vector (3 downto 0);
          Clk, Rst: in std_logic;
          Q_out: out std_logic_vector (3 downto 0));
end REG_4;

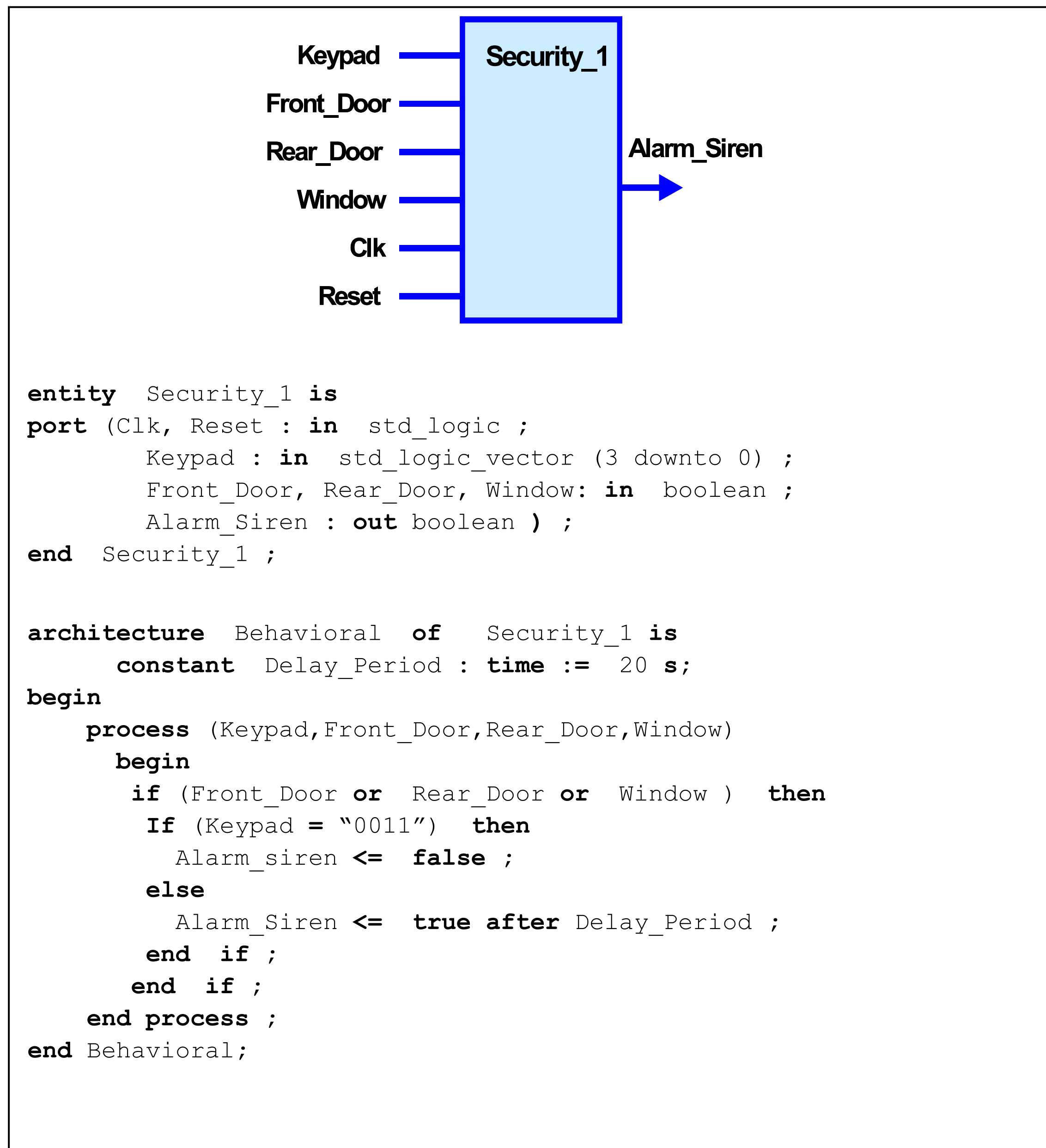
architecture Structural of REG_4 is
      component DFF
          port ( D,Clock : in  std_logic;
                 Reset : in  std_logic;
                 Q : out  std_logic);
      end component;
begin
-- Ánh xạ theo vị trí:
U3:DFF port map(D_in(3), Clk, Rst, Q_out(3));
U2:DFF port map(D_in(2), Clk, Rst, Q_out(2));
U1:DFF port map(D_in(1), Clk, Rst, Q_out(1));
-- Ánh xạ theo tên:
U0:DFF port map(Clock =>Clk, D =>D_in(0),
               Reset =>Rst,Q =>Q_out(0));
```


9.3.2. Phương pháp mô tả theo mô hình hành vi (Behavioral):

Đây là mức độ mô tả trừu tượng nhất, chủ yếu là mô tả theo chức năng của hệ thống số theo yêu cầu đầu vào và đáp ứng ra sử dụng các cấu trúc lệnh như của ngôn ngữ lập trình bậc cao như PROCESS , WAIT, IF, CASE, FOR-LOOP… Mô tả theo cách này tính ngữ nghĩa tự nhiên và giải thuật rất cao, nhập thiết kế rất nhanh, nhưng cấu trúc của phần cứng thường không rõ. Tuy nhiên với những hệ thống phức tạp, yêu cầu cần thiết kế nhanh, mà không cần yêu cầu về mức độ tối ưu phần cứng cao thường dùng cách mô tả này. Người thiết kế chỉ mô tả chức năng, hành vi được mong đợi của thiết kế bằng cách sử dụng mô tả dạng văn bản và các phần tử đồ thị. Phương pháp mô tả này thường dùng cho mô phỏng.

Ví dụ mô tả hệ thống cảnh báo theo mô hình hành vi. Hệ thống gồm có đầu vào từ các sensor (**Front_Door, Rear_Door, Window**), đầu vào từ bàn phím bấm Keypad, tín hiệu **Clk, Reset** và đầu ra điều khiển còi báo động **Alarm_Siren** .

Chức năng hoạt động của hệ thống như sau: Nếu mỗi khi có một sensor nào đó được kích hoạt, thì hệ thống kiểm tra mã bàn phím. Nếu sau 20 giây mà không có mã bàn phím nhập đúng nhập vào thì còi báo động sẽ được bật lên.




```
entity  Security_1 is
port (Clk, Reset : in  std_logic ;
        Keypad : in  std_logic_vector (3 downto 0) ;
        Front_Door, Rear_Door, Window: in  boolean ;
        Alarm_Siren : out boolean ) ;
end  Security_1 ;

architecture  Behavioral  of   Security_1 is
      constant  Delay_Period : time :=  20 s;
begin
    process  (Keypad,Front_Door,Rear_Door,Window)
      begin
       if (Front_Door or  Rear_Door or  Window )   then
        If (Keypad = "0011")   then
          Alarm_siren <=  false ;
        else
          Alarm_Siren <=  true after Delay_Period ;
        end  if ;
       end  if ;
    end process ;
end Behavioral;
```

9.3.3 Phương pháp mô tả theo mô hình luồng dữ liệu RTL

Hệ thống được biểu diễn theo mô hình RTL bao gồm tập các thanh ghi và các phép toán được thực hiện trên dữ liệu số nhị phân được lưu trong các thanh ghi. Luồng dữ liệu và việc xử lý dữ liệu thực hiện trên số liệu được chứa trong các thanh ghi được coi như là hoạt động chuyển đổi giữa các thanh ghi. Ví dụ mô hình RTL này được sử dụng để biểu diễn cấu trúc bộ vi xử lý. Hệ thống số được biểu diễn theo mô hình RTL khi chúng được xác định bởi 3 thành phần như sau:

- Tập các thanh ghi trong hệ thống.
- Các phép toán được thực hiện trên dữ liệu được lưu trong các thanh ghi.
- Những điều khiển để giám sát chuỗi tuần tự các phép toán trong hệ thống.

Thanh ghi gồm nhóm các Trigơ chứa dữ liệu nhị phân và có khả năng thực hiện một hoặc nhiều phép toán cơ bản. Một thanh ghi có thể nạp thông tin mới, dịch thông tin… Một bộ đếm được coi như là một thanh ghi có khả năng tăng, giảm giá trị tuần tự. Một Trigơ có thể coi như là thanh ghi 1 bit. Phần mạch gồm có các Trigơ và các cổng liên quan trong bất cứ mạch tuần tự nào có thể được gọi là những thanh ghi.

Các phép toán được thực hiện trên dữ liệu chứa trong các thanh ghi là những phép toán cơ bản có thể được thực hiện song song trên chuỗi bit trong một chu kỳ clock. Kết quả của phép toán có thể thay thế dữ liệu trước đó của thanh ghi, hoặc kết quả có thể được chuyển đến thanh ghi khác. Có 4 kiểu phép toán như sau:

+ Phép chuyển đổi: truyền dữ liệu từ thanh ghi này sang thanh ghi khác.

+ Phép toán số học.

+ Phép toán logic.

+ Phép dịch.

Điều khiển khởi tạo chuỗi các phép toán bao gồm tín hiệu định thời cho phép thực hiện tuần tự các phép toán theo cách đã được mô tả trước. Có thể coi mô hình RTL là mô hình mô tả hành vi theo từng xung clock của hệ thống số.

Hệ thống số được mô tả bằng VHDL theo mô hình RTL có khả năng tổng hợp rất cao và rất dễ dàng trong việc trao đổi giữa các công cụ tổng hợp, thiết kế, và có thể tổng hợp trên các công nghệ PLD khác nhau.

Theo mô hình RTL, hệ thống số được mô tả bằng các tiến trình tổ hợp (combinatorial process) và các tiến hoạt động theo clock (clocked process)

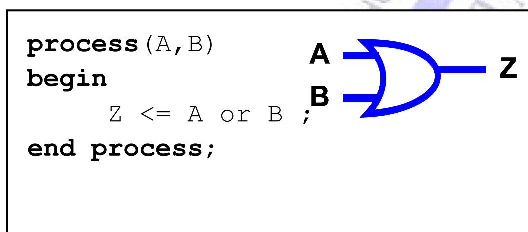
Combinatorial process

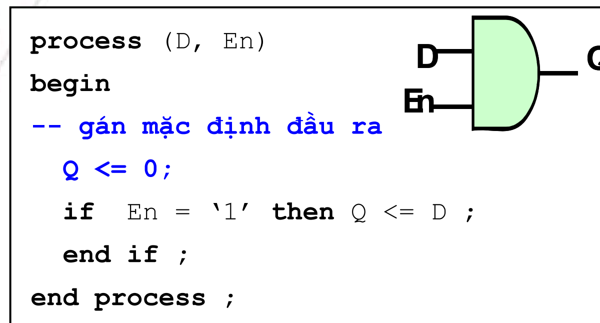
Clocked process

9.3.3.1. Mô tả mạch tổ hợp

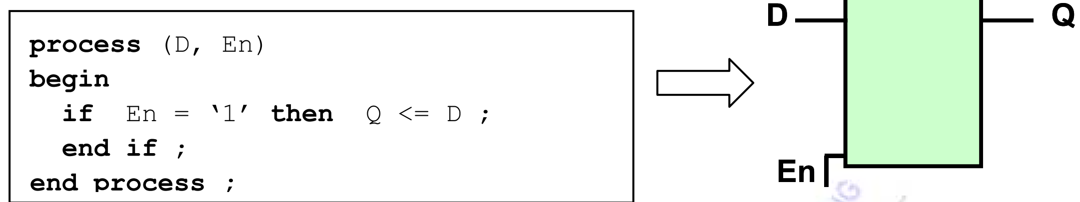
Mạch logic tổ hợp có thể mô tả bằng các cấu trúc lệnh song, tuy nhiên thường dùng các **process** tổ hợp. Trong các process tổ hợp ***tất cả các tín hiệu vào*** của mạch tổ hợp phải được đưa vào danh sách tín hiệu kích thích.

Ví dụ tiến trình tổ hợp như sau:





Chú ý trong các process tổ hợp nên có phép gán giá trị mặc định cho đầu ra để tránh trường hợp mạch bị biến thành mạch chốt theo mức.




```
process (D, En)
begin
  if  En = '1' then  Q <= D ;
  end if ;
end process ;
```

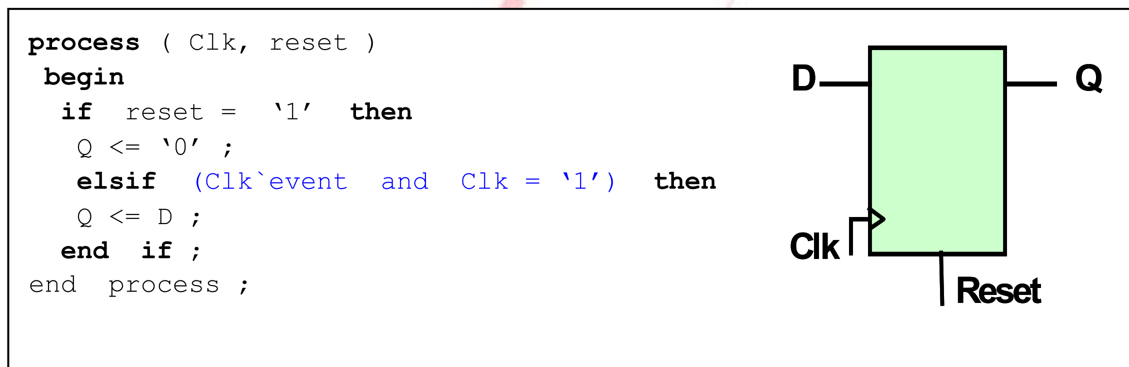
Khi mô tả mạch logic tổ hợp các biến và tín hiệu trong một process không được nhận giá trị khởi tạo trước bởi vì mạch tổ hợp không chứa các phần tử nhớ. Khi trong mô hình mạch các biến hoặc tín hiệu được khởi tạo giá trị trước, chương trình tổng hợp sẽ tạo ra các phần tử nhớ để lưu trữ các giá trị khởi tạo, mạch trở thành mạch có nhớ.

Mọi câu lệnh tuần tự trừ các lệnh wait, loop, if với những tín hiệu điều khiển theo sườn đều có thể dùng để mô tả các mạch tổ hợp. Các phép toán số học, logic, quan hệ đều có thể được sử dụng trong biểu thức.

9.3.3.2. Mô tả mạch tuần tự:

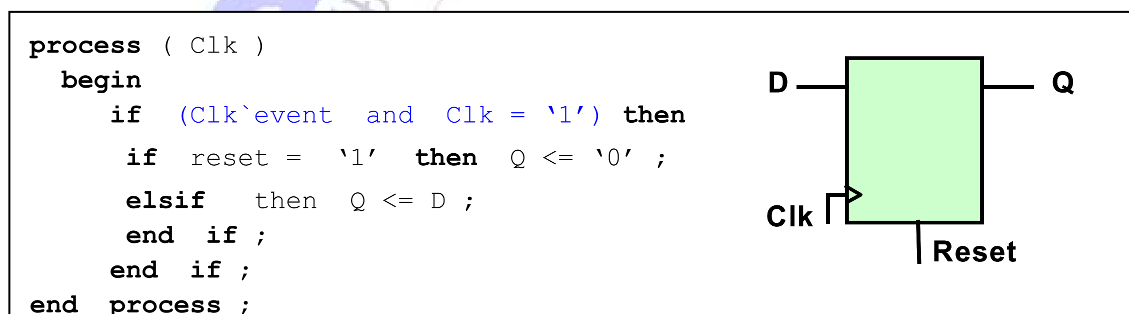
Tiến trình hoạt động theo clock có thể được mô tả thành tiến trình đồng bộ (danh sách tín hiệu kích thích chỉ có duy nhất tín hiệu clock, mọi biến đổi của mạch được đồng bộ theo sườn clock) hoặc thành tiến trình không đồng bộ.

Ví dụ mô tả hoạt động của Triger D làm việc theo sườn dương với các tín hiệu Reset không đồng bộ như sau:



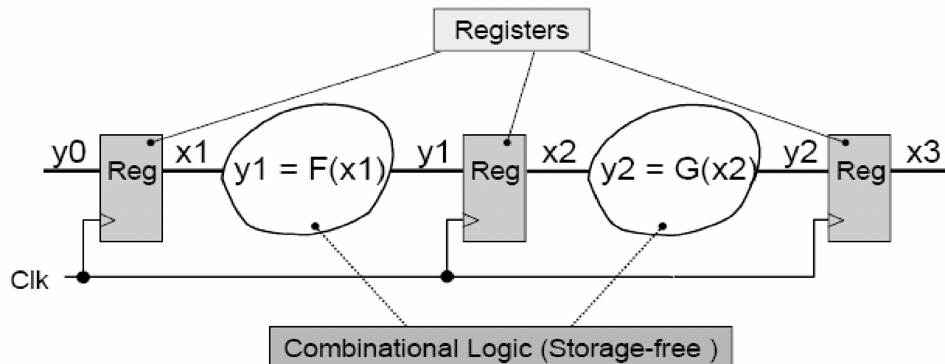

```
process ( Clk, reset )
 begin
  if  reset =  '1'  then
   Q <= '0' ;
   elsif  (Clk`event  and  Clk = '1')  then
   Q <= D ;
  end  if ;
end  process ;
```

Ví dụ mô tả hoạt động của Triger D làm việc theo sườn dương với các tín hiệu Reset đồng bộ như sau:




```
process ( Clk )
  begin
     if  (Clk`event  and  Clk = '1') then
      if  reset =  '1'  then  Q <= '0' ;
      elsif  then  Q <= D ;
      end  if ;
     end  if ;
end  process ;
```

Tóm lại biểu diễn hệ thống số theo mô hình RTL cần sử dụng các cấu trúc thanh ghi (Registers) và mạch tổ hợp (combinational logic), ví dụ tả datapath theo mô hình RTL như hình vẽ 9-5 sau:



Hình 9-5. Một mô hình RTL

Mô tả VHDL cho mô hình trên có thể thực hiện theo 2 cách như sau:

```
architecture SPLIT of DATAPATH is
      signal X1, Y1, X2, Y2 : ...
begin
     REG : process (CLK)
     begin
     if (CLK'event and CLK = '1') then
          X1 <= Y0;
          X2 <= Y1;
          X3 <= Y2;
     end if;
     end process;

     LOGIC : process (X1, X2)
     begin
          Y1 <= F(X1);
          Y2 <= G(X2);
     end process;
end SPLIT;
```

Registe rs

```
architecture COMBINED of DATAPATH is
      signal X1, X2 : ...
begin
     process (CLK)                                  -- Registers
     begin
     if (CLK'event and CLK = '1') then
          X2 <= F(X1);                      }  Combinational Logic
          X3 <= G(X2);
          X1 <= Y0;
     end if;
     end process;
```

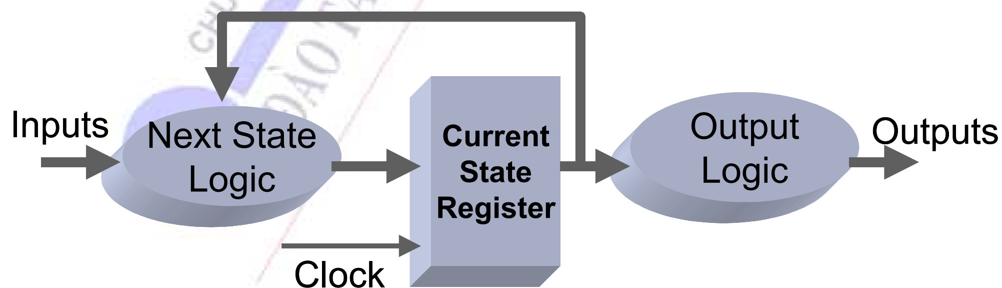
9.3.4 Phương pháp mô tả theo mô hình đồ hình trạng thái (máy trạng thái State Machine)

Hoạt động của một hệ thống số tuần tự có thể được mô tả dưới dạng đồ hình trạng thái Moore hoặc Mealy. Dùng VHDL có thể mô tả được đồ hình chuyển đổi trạng thái đó. Bảng sau cho biết khả năng mô tả đồ hình trạng thái dùng VHDL:

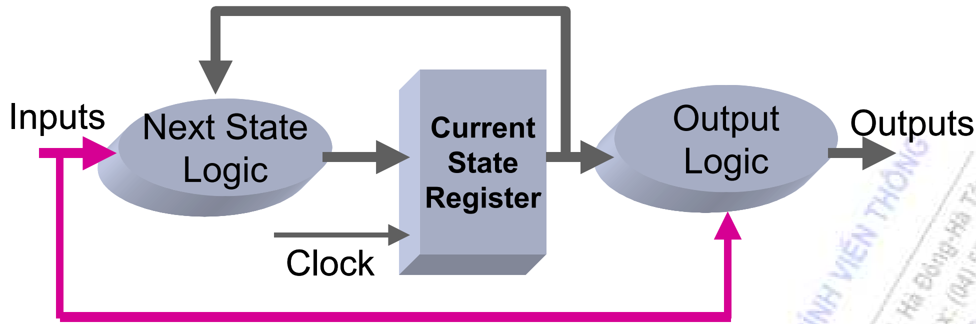
STT	Yêu cầu mô tả	Sử dụng cấu trúc trong VHDL
1	- Trạng thái logic hiện tại	- **Process** hoạt động theo clock
2	- Xác định trạng thái logic tiếp theo	- **Process** tổ hợp
3	- Xác định đầu ra	- **Process** tổ hợp
4	- Đặt tên cho các trạng thái	- Kiểu dữ liệu liệt kê
5	- Đánh giá mỗi trạng thái	- Lệnh **Case**
6	- Đánh giá các điều kiện đầu vào	- Lệnh **if/else**

Tổng kết lại các kiểu đồ hình trạng thái như sau:

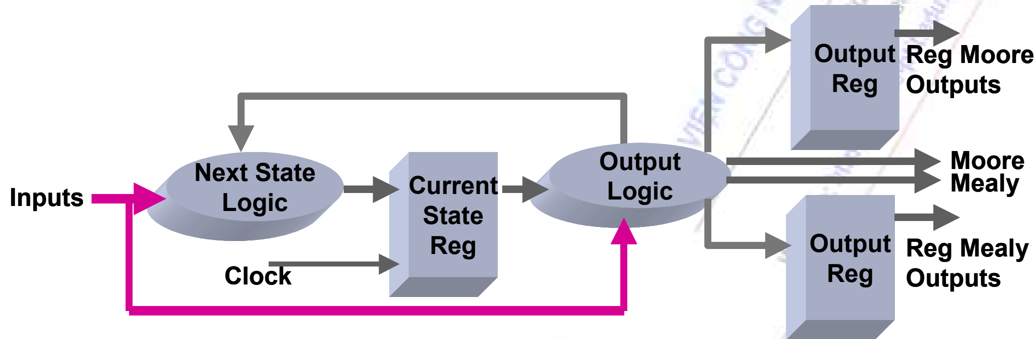
- Mô hình Moore: Kết quả đầu ra chỉ phụ thuộc vào trạng thái hiện tại.



- **Mô hình Mealy**: Đầu ra phụ thuộc vào cả trạng thái hiện tại và tín hiệu vào.



Trong thực tế hệ thống số thường được mô tả bằng việc kết hợp cả mô hình Moore và Mealy:



- Cách sử dụng kiểu dữ liệu liệt kê để đặt tên cho các trạng thái như sau:

```
architecture RTL of FSM is

. . .
type My_State is ( Init, Load, Fetch, Stor_A, Stor_B) ;
signal Current_State, Next_State : My_State;
. . .
begin
```

- Cách sử dụng hằng để mã hóa các trạng theo như mong muốn:

```
subtype My_State is std_logic_vector( 0 to 5 ) ;

constant  Init       : My_State := "111000" ;
constant  Load       : My_State := "101010" ;
constant  Init       : My_State := "000011" ;
signal  Curr_State, Next_State : My_State ;
. . .
begin --architecture
```

- Để mô tả quá trình chuyển đổi trạng thái và cập nhật kết quả đầu ra ứng với mỗi trạng thái thông thường sử dụng cách mô tả bằng nhiều tiến trình

+ Tiến trình cập nhập trạng thái mới của hệ thống (tiến trình Sync).

```
Sync: process  ( CLK , RST)
  begin
       . . .
end process Sync ;
```

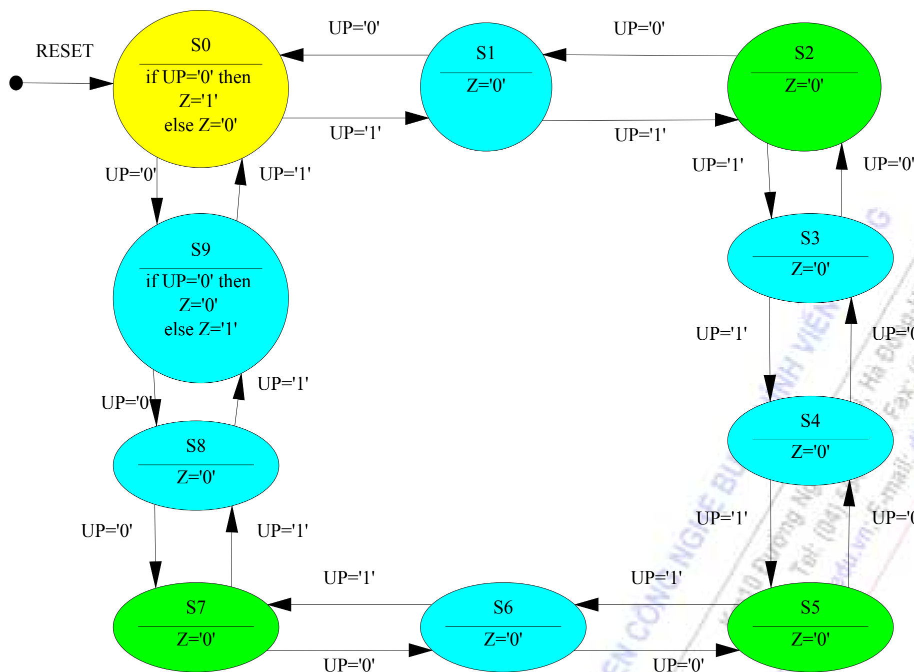
+ Tiến trình kiểm tra điều kiện chuyển đổi trạng thái (tiến trình Comb).

```
 Comb: process  ( Curr_State, In1, In2…)
  begin
       . . .
end process Comb ;
```

+ Tiến trình cập kết quả đầu ra ứng với mỗi trạng thái (tiến trình Outputs).

```
Outputs: process  ( Curr_State, In1, In2…)
  begin
       . . .
end process Outputs ;
```

- Ví dụ bộ đếm thập phân thuận nghịch đồng bộ có đồ đồ hình trạng thái như sau:



- Mô tả VHDL cho đồ hình trạng thái trên như sau:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY FSM IS
      PORT (CLK,RESET,UP: IN std_logic;
                        Z : OUT std_logic);
END;
ARCHITECTURE BEHAVIOR OF FSM IS
      SIGNAL sreg : std_logic_vector (3 DOWNTO 0);
      SIGNAL next_sreg : std_logic_vector (3 DOWNTO 0);
      CONSTANT S0 : std_logic_vector (3 DOWNTO 0) :="0000";
      CONSTANT S1 : std_logic_vector (3 DOWNTO 0) :="0001";
      CONSTANT S2 : std_logic_vector (3 DOWNTO 0) :="0010";
      CONSTANT S3 : std_logic_vector (3 DOWNTO 0) :="0011";
      CONSTANT S4 : std_logic_vector (3 DOWNTO 0) :="0100";
      CONSTANT S5 : std_logic_vector (3 DOWNTO 0) :="0101";
      CONSTANT S6 : std_logic_vector (3 DOWNTO 0) :="0110";
      CONSTANT S7 : std_logic_vector (3 DOWNTO 0) :="0111";
      CONSTANT S8 : std_logic_vector (3 DOWNTO 0) :="1000";
      CONSTANT S9 : std_logic_vector (3 DOWNTO 0) :="1001";

      SIGNAL next_Z : std_logic;
BEGIN
      Sync: PROCESS (CLK)
      BEGIN
            IF CLK='1' AND CLK'event THEN
```

```
          if RESET='1' then
            sreg<= S0;
          else
             sreg <= next_sreg;
          end if;
       END IF;
END PROCESS;

Comb: PROCESS (sreg,UP)
BEGIN
    CASE sreg IS
         WHEN S0 =>
             IF ( UP='0' ) THEN      next_sreg<=S9;
             ELSE              next_sreg<=S1;
             END IF;
         WHEN S1 =>
             IF ( UP='0' ) THEN      next_sreg<=S0;
             ELSE              next_sreg<=S2;
             END IF;
        WHEN S2 =>
             IF ( UP='0' ) THEN      next_sreg<=S1;
             ELSE              next_sreg<=S3;
             END IF;
         WHEN S3 =>
             IF ( UP='0' ) THEN      next_sreg<=S2;
             ELSE              next_sreg<=S4;
             END IF;
         WHEN S4 =>
             IF ( UP='0' ) THEN      next_sreg<=S3;
             ELSE              next_sreg<=S5;
             END IF;
         WHEN S5 =>
             IF ( UP='0' ) THEN      next_sreg<=S4;
             ELSE              next_sreg<=S6;
             END IF;
         WHEN S6 =>
             IF ( UP='0' ) THEN      next_sreg<=S5;
             ELSE              next_sreg<=S7;
             END IF;
         WHEN S7 =>
             IF ( UP='0' ) THEN      next_sreg<=S6;
             ELSE              next_sreg<=S8;
```

```
                END IF;
            WHEN S8 =>
                IF ( UP='0' ) THEN      next_sreg<=S7;
                ELSE                next_sreg<=S9;
                END IF;
            WHEN S9 =>
                IF ( UP='0' ) THEN      next_sreg<=S8;
                ELSE                next_sreg<=S0;
                END IF;
            WHEN OTHERS => next_sreg<=S0;
        END CASE;
    END PROCESS;
    Outputs: PROCESS (sreg,UP)
    BEGIN
          IF UP='1' THEN
             if sreg=S9 then   Z<= '1';
             else              Z<= '0';
             end if;
          ELSE
             if sreg=S0 then   Z<= '1';
             else              Z<= '0';
             end if;
          END IF;
    END PROCESS;
END BEHAVIOR;
```

TÓM TẮT

Thiết kế với sự trợ giúp của máy tính của các hệ thống kỹ thuật số được dùng rộng rãi trong công nghiệp. Do đó, ta cần phải hiểu các khái niệm khác nhau trong quá trính thiết kế. Ngôn ngữ mô tả phần cứng phổ biến VHDL là loại ngôn ngữ được trình bày trong chương này. Đây là một chủ đề rất rộng nên chúng tôi khổng thể trình bày chi tiết của VHDL. Tuy nhiên các khái niệm cơ bản đọc trình bày ở đây sẽ giúp cho chúng ta học những chi tiết về ngôn ngữ từ những quyển sách viết về VHDL

CÂU HỎI ÔN TẬP CHƯƠNG 8 VÀ CHƯƠNG 9

1. Đặc điểm nào dưới đây là nhược điểm của phương pháp thiết kế mạch dùng IC có chức năng cố định?
   A. Chi phí thiết kế thấp
   B. Vận hành nhanh xung quanh bản thiết kế
   C. Khó khăn khi triển khai các thiết kế phức tạp
   D. Tương đối dễ dàng khi thử nghiệm các mạch thiết kế

2. Đặc điểm nào dưới đây là ưu điểm của phương pháp thiết kế mạch dùng IC có chức năng cố định?
   A. Yêu cầu công suất điện tiêu thụ lớn
   B. Khó khăn khi sửa chữa, nâng cấp thiết kế
   C. Thiếu tính bảo mật
   D. Tương đối dễ dàng khi thử nghiệm mạch thiết kế

3. Trong số các loại cấu kiện logic sau, loại nào không thuộc họ PLD
   A. CPLD
   B. FPGA
   C. Vi xử lý
   D. SPLD

4. Đặc điểm nào dưới đây không phải là ưu điểm của PLD
   A. Mật độ tích hợp cao.
   B. Bảo đảm tính bảo mật của thiết kế
   C. Thời gian thiết kế ngắn
   D. Chi phí sản xuất số lượng lớn cao

5. Trong cấu trúc của SPLD không có phần tử nào
   A. Mảng các cổng logic AND,OR.
   B. Ma trận kết nối
   C. Bộ nhớ RAM
   D. Triger

6. Khối nào sau đây không có trong cấu trúc của CPLD
   A. Khối logic gồm ma trận hạng tích AND, OR
   B. Khối Microcell chứa tài nguyên về các Triger, thanh ghi

C. Ma trận kết nối trung tâm

D. Vi xử lý

7. Để thực hiện hàm logic tổ hợp trong FPGA sử dụng

A. Ma trận hạng tích AND, OR.

B. Cấu trúc bảng tra LUT dựa vào SDRAM .

C. Các cấu trúc thanh ghi

D. Cấu trúc vào/ra.

8. Xác định phát biểu sai trong số các phát biểu sau

A. FPGA có cấu trúc không đồng nhất

B. CPLD có cấu trúc đồng nhất

C. Cấu hình của CPLD được lưu lại khi mất điện

D. Cấu hình trong FPGA dựa vào công nghệ SRAM được lưu lại khi mất điện

9. Trình tự thực hiện trong lưu đồ thiết kế cho CPLD/FPGA là:

A. Nhập thiết kế, kiểm tra thiết kế, tổng hợp thiết kế,mô phỏng định thời, thực hiện thiết kế, cấu hình.

B. Nhập thiết kế, kiểm tra thiết kế, thực hiện thiết kế, tổng hợp thiết kế, mô phỏng định thời, cấu hình.

C. Nhập thiết kế, tổng hợp thiết kế, kiểm tra thiết kế, thực hiện thiết kế, mô phỏng định thời, cấu hình.

D. Nhập thiết kế, kiểm tra thiết kế, tổng hợp thiết kế, thực hiện thiết kế, mô phỏng định thời, cấu hình.

10. Kết quả của bước tổng hợp thiết kế trong lưu đồ thiết kế cho CPLD/FPGA là:

A. File mô tả VHDL

B. File cấu hình

C. File netlist

D. File sơ đồ mạch

11. Kết quả của bước thực hiện thiết kế trong lưu đồ thiết kế cho CPLD/FPGA là:

A. File mô tả VHDL

B. File cấu hình

C. File netlist

D. File sơ đồ mạch

12. Trong bước thực hiện thiết kế của lưu đồ thiết kế cho CPLD/FPGA gồm các chức năng:

A. Mô phỏng chức năng, tổng hợp thiết kế.
B. Biên dịch, map, Định vị trí và định tuyến kế nối.
C. Mô phỏng định thời, tạo cấu hình, biên dịch.
D. Tạo file mô tả HDL, tổng hợp thiết kế, Định vị trí và định tuyến kế nối.

13. VHDL là ngôn ngữ:
A. Lập trình hợp ngữ
B. Lập trình bậc cao
C. Lập trình mạng
D. Mô tả phần cứng

14. Trình tự sắp xếp theo mức độ mô tả trừu tượng tăng dần dùng VHDL là:
A. Mức hành vi, mức luồng dữ liệu RTL, mức logic, mức layout.
B. Mức hành vi, mức logic, mức luồng dữ liệu RTL, mức layout.
C. Mức layout, mức logic, mức hành vi, mức luồng dữ liệu RTL.
D. Mức layout, mức logic, mức luồng dữ liệu RTL, mức hành vi.

15. Đối tượng tín hiệu (signal) trong ngôn ngữ VHDL để :
A. Lưu các kết quả trung gian
B. Biểu diễn đường kết nối trong hệ thống phần cứng số
C. Lưu những giá trị cố định
D. Biểu diễn cổng vào hoặc ra của thực thể

16. Đối tượng biến (variable) trong ngôn ngữ VHDL để :
A. Lưu các kết quả trung gian
A. Biểu diễn đường kết nối trong hệ thống phần cứng số
C. Lưu những giá trị cố định
D. Biểu diễn cổng vào hoặc ra của thực thể

17. Cho khai báo của các đối tượng như sau:

signal A : in std_logic;

Phép gán nào đúng:
A. A:='1';
B. A<=1;
C. A<='1';
D. A<=true;

18. Cho khai báo của các đối tượng như sau:

    Variable A : in std_logic;

    Phép gán nào đúng:

    A. A<=true;

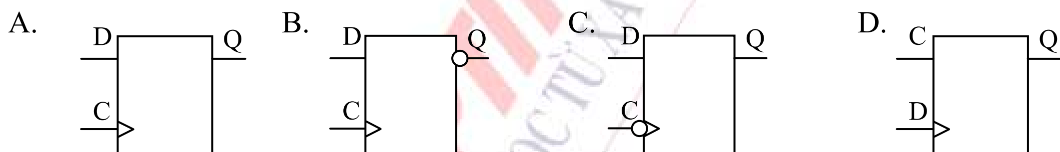
    B. A:=1;

    C. A<='1';

    D. A:='1';

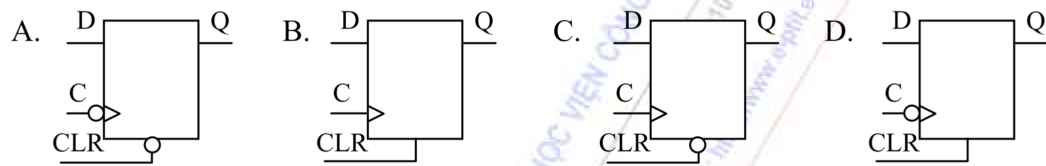
19. Mô hình phần cứng nào tổng hợp được ứng với đoạn mô tả như sau:

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port(C, D : in std_logic;
          Q : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
          Q <= D;
        end if;
    end process;
end archi;
```



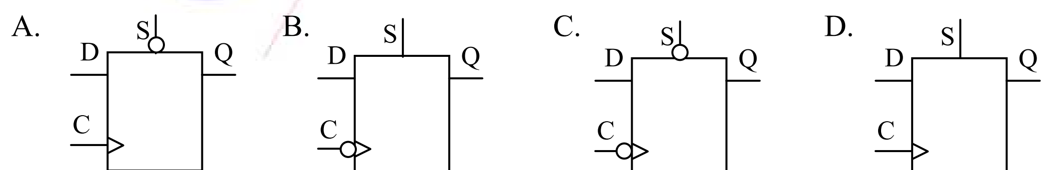
20. Mô hình phần cứng nào tổng hợp được ứng với đoạn mô tả như sau:

```
entity flop is
  port(C, D, CLR : in std_logic;
        Q        : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C, CLR)
      begin
        if (CLR = '1')then
          Q <= '0';
        elsif (C'event and C='0')then
          Q <= D;
        end if;
    end process;
end archi;
```



21. Mô hình phần cứng nào tổng hợp được ứng với đoạn mô tả như sau:

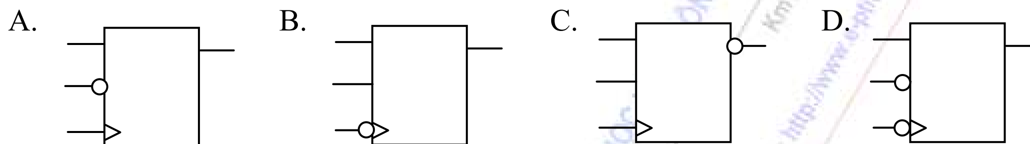
```
entity flop is
  port(C, D, S  : in  std_logic;
        Q       : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (S='1') then
            Q <= '1';
          else
            Q <= D;
          end if;
        end if;
    end process;
end archi;
```



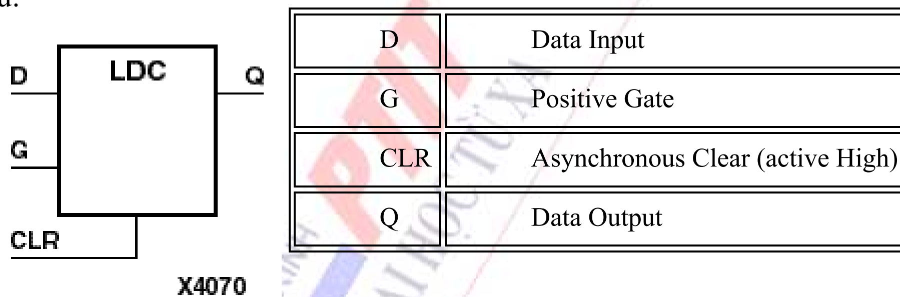
22. Mô hình phần cứng nào tổng hợp được ứng với đoạn mô tả như sau:

```
entity flop is
  port(C, D, CE : in  std_logic;
             Q : out std_logic);
end flop;
architecture archi of flop is
  begin
    process (C)
      begin
        if (C'event and C='1') then
          if (CE='0') then
            Q <= D;
          end if;
        end if;
    end process;
end archi;
```

A. B. C. D.



23. Đoạn mô tả VHDL nào mô tả cho mô hình mạch chốt cổng dương và xóa không đồng bộ như sau:



D	Data Input
G	Positive Gate
CLR	Asynchronous Clear (active High)
Q	Data Output

A.	B.
entity latch is port(G, D, CLR : in std_logic; Q : out std_logic); end latch; architecture archi of latch is begin process (CLR, D, G) begin if (CLR='1') then Q <= '1'; elsif (G='1') then Q <= D; end if; end process; end archi;	entity latch is port(G, D, CLR : in std_logic; Q : out std_logic); end latch; architecture archi of latch is begin process (CLR, D, G) begin if (CLR='0') then Q <= '0'; elsif (G='1') then Q <= D; end if; end process; end archi;

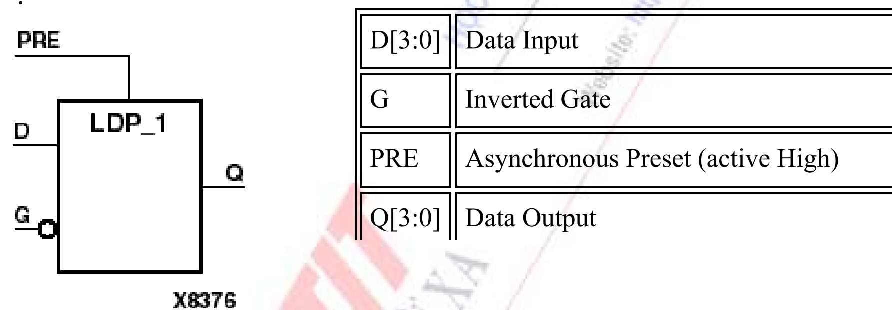
C.

```
entity latch is
  port(G, D, CLR : in  std_logic;
         Q : out std_logic);
end latch;
architecture archi of latch is
  begin
    process (CLR, D, G)
      begin
        if (CLR='1') then
          Q <= '0';
        elsif (G='1') then
          Q <= D;
        end if;
    end process;
end archi;
```

D.

```
entity latch is
  port(G, D, CLR : in  std_logic;
         Q : out std_logic);
end latch;
architecture archi of latch is
  begin
    process (CLR, D, G)
      begin
        if (CLR='1') then
          Q <= '0';
        elsif (G='0') then
          Q <= D;
        end if;
    end process;
end archi;
```

24. Đoạn mô tả kiến trúc nào mô tả cho mô hình mạch chốt cổng đảo và Preset không đồng bộ như sau:



D[3:0]	Data Input
G	Inverted Gate
PRE	Asynchronous Preset (active High)
Q[3:0]	Data Output

Trong đó mô tả thực thể như sau:

```
entity latch is
  port(D : in std_logic_vector(3 downto 0);
         G, PRE : in  std_logic;
         Q  : out std_logic_vector(3 downto 0));
end latch;
```

A.

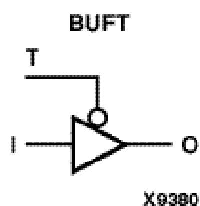
```
architecture archi of latch is
  begin
    process (PRE, G)
      begin
        if (Q='1') then
          Q <= "1111";
        elsif (PRE='0') then
          Q <= D;
        end if;
    end process;
end archi;
```

B.

```
architecture archi of latch is
  begin
    process (PRE, G)
      begin
        if (PRE='1') then
          Q <= "1111";
        elsif (G='0') then
          Q <= D;
        end if;
    end process;
end archi;
```

C.	D.
architecture archi of latch is begin process (PRE) begin if (PRE='1') then Q <= "1111"; elsif (G='0') then Q <= D; end if; end process; end archi;	architecture archi of latch is begin process (PRE, G) begin if (PRE='1') then Q <= "1111"; elsif (G='1') then Q <= D; end if; end process; end archi;

25. Đoạn mô tả kiến trúc nào mô tả cho cổng 3 trạng thái sau:



Trong đó mô tả thực thể như sau:

```
entity three_st is
  port( T, I : in  std_logic;
             O : out std_logic);
end three_st;
```

A.	B.
architecture archi of three_st is begin process (I, T) begin if (T='0') then O <= I; else O <= 'X'; end if; end process; end archi;	architecture archi of three_st is begin process (I, T) begin if (T='1') then O <= I; else O <= 'Z'; end if; end process; end archi;
C.	**D.**
architecture archi of three_st is begin O <= I when T='1' else 'Z'; end archi;	architecture archi of three_st is begin O <= I when T='0' else 'Z'; end archi;

26. Đoạn mô tả kiến trúc nào mô tả hoạt động của bộ đếm tiến 4 bit có xóa không đồng bộ có mô tả thực thể như sau:

```
entity counter is
  port( Clk, CLR : in  std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
```

<table>
<tr>
<td>
A.
<pre>
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk, CLR)
  begin
  if (CLR='1') then
            tmp <= "0000";
  elsif (Clk'event and Clk='1')
then
            tmp <= tmp + 1;
  end if;
 end process;
   Q <= tmp;
end archi;
</pre>
</td>
<td>
B.
<pre>
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk)
  begin
  if (Clk'event and Clk='1')then
   if (CLR='1') then
            tmp <= "0000";
   else tmp <= tmp + 1;
   end if;
  end if;
 end process;
   Q <= tmp;
end archi;
</pre>
</td>
</tr>
<tr>
<td>
C.
<pre>
architecture archi of counter is
 begin
 process (Clk, CLR)
  begin
  if (CLR='1') then
            Q <= "0000";
  elsif(Clk'event and Clk='0')
  then
        Q <= Q + 1;
  end if;
 end process;
end archi;
</pre>
</td>
<td>
D.
<pre>
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk)
  begin
  if (Clk'event and Clk='0')then
   if (CLR='1') then
            tmp <= "0000";
   else tmp <= tmp - 1;
   end if;
  end if;
 end process;
   Q <= tmp;
end archi;
</pre>
</td>
</tr>
</table>

27. Mô hình mạch số nào có mô tả VHDL như sau:

```
entity counter is
  port( Clk, S : in  std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (Clk)
      begin
        if (Clk'event and Clk='1') then
          if (S='1') then
            tmp <= "1111";
          else
            tmp <= tmp - 1;
          end if;
        end if;
    end process;
    Q <= tmp;
end archi;
```

A. Bộ đếm lùi 4 bit ra Q[3:0], hoạt động ở sườn âm của clock CLK, tín hiệu thiết lập S tích cực dương và đồng bộ

B. Bộ đếm lùi 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, tín hiệu thiết lập S tích cực dương và đồng bộ.

C. Bộ đếm tiến 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, tín hiệu thiết lập S tích cực dương và đồng bộ

D. Bộ đếm lùi 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, tín hiệu thiết lập S tích cực dương và không đồng bộ.

28. Đoạn mô tả kiến trúc nào mô tả hoạt động của bộ đếm tiến 4 bit nạp không đồng bộ từ tín hiệu đầu vào, hoạt động ở sườn clock âm và có mô tả thực thể như sau:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port( Clk, ALOAD : in  std_logic;  -- Clock và tín hiệu nạp
        D : in std_logic_vector(3 downto 0); -- Đầu vào bộ đếm
        Q : out std_logic_vector(3 downto 0)); -- Đầu ra bộ đếm
end counter;
```

A.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk,ALOAD, D)
  begin
   if (ALOAD='1') then
     tmp <= D;
   elsif (Clk'event and Clk='1')
   then tmp <= tmp + 1;
   end if;
 end process;
    Q <= tmp;
end archi;
```

B.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk,D)
  begin
   if (ALOAD='1') then
     tmp <= D;
   elsif (Clk'event and Clk='0')
   then tmp <= tmp + 1;
   end if;
 end process;
    Q <= tmp;
end archi;
```

C.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk,ALOAD, D)
  begin
   if (ALOAD='1') then
     tmp <= D;
   elsif (Clk'event and Clk='0')
   then tmp <= tmp + 1;
   end if;
 end process;
    Q <= tmp;
end archi;
```

D.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                          downto 0);
begin
 process (Clk)
  begin
   if (ALOAD='1') then
     tmp <= D;
   elsif (Clk'event and Clk='0')
   then tmp <= tmp + 1;
   end if;
 end process;
    Q <= tmp;
end archi;
```

29. Mô hình mạch số nào có mô tả VHDL như sau:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port( Clk, SLOAD : in  std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
   begin
    process (Clk)
      begin
        if (Clk'event and Clk='1') then
          if (SLOAD='1') then
            tmp <= "1010";
          else
            tmp <= tmp + 1;
          end if;
        end if;
    end process;
    Q <= tmp;
end archi;
```

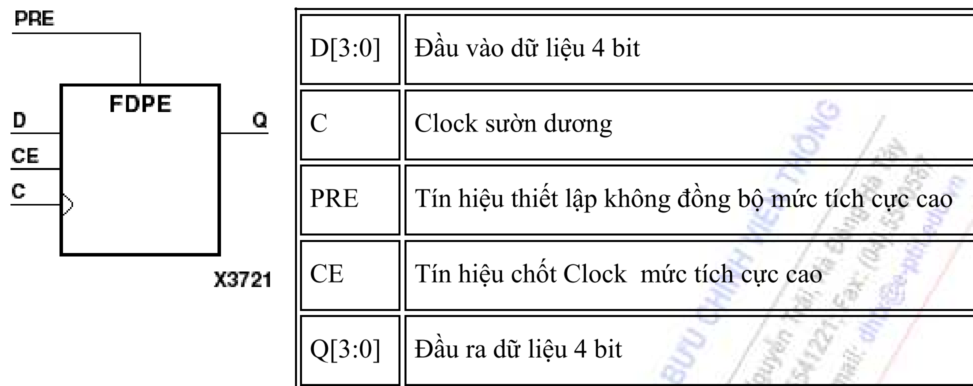
A. Bộ đếm tiến 4 bit ra Q[3:0], hoạt động ở sườn âm của clock CLK, nạp đồng bộ hằng số "1010" (theo mức tích cực dương).

B. Bộ đếm tiến 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, nạp không đồng bộ hằng số "1010" (theo mức tích cực dương).

C. Bộ đếm tiến 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, nạp đồng bộ hằng số "1010" (theo mức tích cực dương).

D. Bộ đếm lùi 4 bit ra Q[3:0], hoạt động ở sườn dương của clock CLK, nạp đồng bộ hằng số "1010" (theo mức tích cực dương).

30. Đoạn mô tả kiến trúc nào mô tả cho mô hình thanh ghi 4 bit hoạt động sườn dươn của clock, có tín hiệu chốt clock và thiết lập không đồng bộ,



D[3:0]	Đầu vào dữ liệu 4 bit
C	Clock sườn dương
PRE	Tín hiệu thiết lập không đồng bộ mức tích cực cao
CE	Tín hiệu chốt Clock mức tích cực cao
Q[3:0]	Đầu ra dữ liệu 4 bit

Mô tả thực thể của thanh ghi như sau:

```
library ieee;
use ieee.std_logic_1164.all;
entity flop is
  port( C, CE, PRE : in std_logic;
        D : in  std_logic_vector (3 downto 0);
        Q : out std_logic_vector (3 downto 0));
end flop;
```

A.

```
architecture archi of flop is
 begin
  process (C)
  begin
   if (PRE='1') then
        Q <= "1111";
   elsif (C'event and C='1')then
         if (CE='1') then
            Q <= D;
         end if;
   end if;
  end process;
end archi;
```

B.

```
architecture archi of flop is
 begin
  process (C, PRE)
  begin
   if (PRE='1') then
        Q <= "1111";
   elsif (C'event and C='1')then
         if (CE='0') then
            Q <= D;
         end if;
   end if;
  end process;
end archi;
```

C.

```
architecture archi of flop is
 begin
  process (C, PRE)
  begin
   if (PRE='1') then
        Q <= "1111";
   elsif (C'event and C='1')then
         if (CE='1') then
            Q <= D;
         end if;
   end if;
  end process;
end archi;
```

D.

```
architecture archi of flop is
 begin
  process (C, PRE)
  begin
   if (PRE='1') then
        Q <= "0000";
   elsif (C'event and C='1')then
         if (CE='1') then
            Q <= D;
         end if;
   end if;
  end process;
end archi;
```

31. Mô hình mạch số nào có đoạn mô tả VHDL như sau:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port( Clk, SLOAD : in  std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
  signal tmp: std_logic_vector(3 downto 0);
   begin
    process (Clk)
      begin
        if (Clk'event and Clk='1') then
          if (SLOAD='1') then tmp <= "1010";
          else    tmp <= tmp + 1;
          end if;
        end if;
    end process;
    Q <= tmp;
end archi;
```

A. Bộ đếm tiến 4 bit đầu ra Q [3:0] hoạt động với sườn âm clock, nạp đồng bộ giá trị cố định "1010" mức tích cực cao

B. Bộ đếm tiến 4 bit đầu ra Q [3:0] hoạt động với sườn dương clock, nạp đồng bộ giá trị cố định "1010" mức tích cực cao

C. Bộ đếm tiến 4 bit đầu ra Q [3:0] hoạt động với sườn dương clock, nạp đồng bộ giá trị cố định "1010" mức tích cực thấp

D. Bộ đếm tiến 4 bit đầu ra Q [3:0] hoạt động với sườn dương clock, nạp không đồng bộ giá trị cố định "1010" mức tích cực cao

32. Đoạn mô tả kiến trúc nào mô tả cho mô hình bộ đếm thuận/nghịch 4 bit có xóa không đồng bộ, có mô tả thực thể như sau:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port( C, CLR, up_down : in std_logic; -- C - clock
        Q : out std_logic_vector(3 downto 0));
end counter;
```

A.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                        downto 0);
begin
 process (C, CLR)
  begin
  if (CLR='1') then
        tmp <= "0000";
  elsif (C'event and C='1') then
    if (up_down='1') then
        tmp <= tmp + 1;
    else tmp <= tmp - 1;
    end if;
  end if;
 end process;
 Q <= tmp;
end archi;
```

B.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                        downto 0);
begin
 process (C)
  begin
  if (CLR='1') then
        tmp <= "0000";
  elsif (C'event and C='1') then
    if (up_down='1') then
        tmp <= tmp + 1;
    else tmp <= tmp - 1;
    end if;
  end if;
 end process;
 Q <= tmp;
end archi;
```

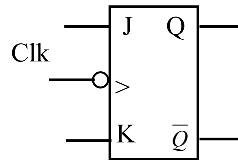
C.

```
architecture archi of counter is
begin
 process (C, CLR)
  begin
  if (CLR='1') then
        Q <= "0000";
  elsif (C'event and C='1') then
    if (up_down='1') then
        Q <= Q + 1;
    else tmp <= tmp - 1;
    end if;
  end if;
 end process;
end archi;
```

D.

```
architecture archi of counter is
  signal tmp: std_logic_vector(3
                     downto 0);
begin
 process (C, CLR)
  begin
  if (CLR='1') then
        tmp <= "1111";
  elsif (C'event and C='1') then
    if (up_down='1') then
        tmp <= tmp + 1;
    else tmp <= tmp - 1;
    end if;
  end if;
 end process;
 Q <= tmp;
end archi;
```

33. Đoạn mô tả nào mô tả cho triger JK sau:



A.

```
entity JKFF is
    Port(J,K,Clk:in std_logic;
         Q, notQ:out std_logic);
end JKFF;
architecture Behavioral of JKFF is
signal Qtemp: std_logic;
signal JK:std_logic_vector(0 to 1);
begin
   JK<=(J,K);
   process(Clk)
   begin
    if(Clk'event and Clk='1') then
      case JK is
      when "00"  =>  Null;
      when "01"  => Qtemp<='0';
      when "10"  => Qtemp<='1';
     when others=>Qtemp<=not Qtemp;
      end case;
    end if;
   end process;
   Q<=Qtemp;
   notQ<=not Qtemp;
end Behavioral;
```

C.

```
entity JKFF is
    Port(J,K,Clk:in std_logic;
         Q, notQ:out std_logic);
end JKFF;
architecture Behavioral of JKFF is
signal Qtemp: std_logic;
signal JK:std_logic_vector(0 to 1);
begin
   JK<=(J,K);
   process(Clk)
   begin
    if(Clk'event and Clk='0') then
      case JK is
      when "00"  =>  Null;
      when "01"  => Qtemp<='0';
      when "10"  => Qtemp<='1';
     when others=>Qtemp<=not Qtemp;
      end case;
    end if;
   end process;
   Q<=Qtemp;
   notQ<=not Qtemp;
end Behavioral;
```

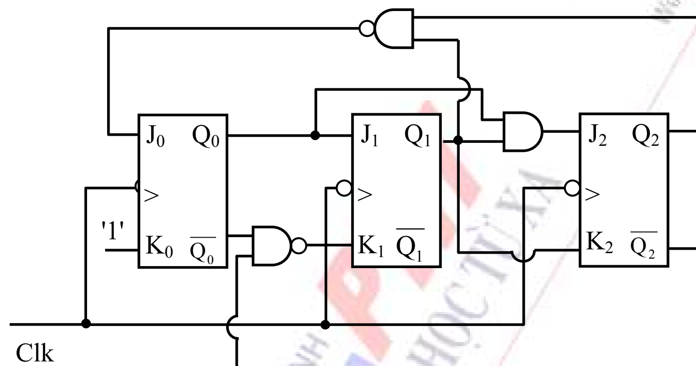
B.

```
entity JKFF is
    Port(J,K,Clk:in std_logic;
         Q, notQ:out std_logic);
end JKFF;
architecture Behavioral of JKFF
is
begin
   process(Clk)
   begin
    if(Clk'event and Clk='1')
then
      Q<=J; notQ<=K;
    end if;
   end process;
end Behavioral;
```

D.

```
entity JKFF is
    Port(J,K,Clk:in std_logic;
         Q, notQ:out std_logic);
end JKFF;
architecture Behavioral of JKFF
is
begin
   process(Clk)
   begin
    if(Clk'event and Clk='0')
then
      Q<=J; notQ<=K;
    end if;
   end process;
end Behavioral;
```

34. Đoạn mô tả nào mô tả đúng cho mạch sau theo mô hình RTL:



A.

```
architecture Behavioral of cau33 is
begin
   notQ<=not Q;
   J(0)<=Q(1) nand Q(2); K(0)<='1';
   J(1)<=Q(0);     K(1)<= notQ(0) nand notQ(2);
   J(2)<=Q(1) and Q(0); K(2)<=Q(1);
end Behavioral;
```

B.

```
architecture Behavioral of cau33 is
 signal Clk: std_logic;
 signal J,K,Q,notQ: std_logic_vector(0 to 2);
 signal JK0,JK1,JK2: std_logic_vector(0 to 1);
begin
  JK0 <=(J(0),K(0));JK1 <=(J(1),K(1));
  JK2 <=(J(2),K(2));
   notQ<=not Q;
   J(0)<=Q(1) nand Q(2); K(0)<='1';
   J(1)<=Q(0);     K(1)<= notQ(0) nand notQ(2);
   J(2)<=Q(1) and Q(0); K(2)<=Q(1);
end Behavioral;
```

C.

```
architecture Behavioral of cau33 is
begin
   process(Clk)
    begin
       if(Clk'event and Clk='1') then
            case JK0 is
                  when "00"   =>  Null;
                  when "01"     => Q(0)<='0';
                  when "10"     => Q(0)<='1';
                  when others => Q(0)<= not Q(0);
            end case;
       end if;
   end process;
end Behavioral;
```

D.

```
architecture Behavioral of cau33 is
 signal Clk: std_logic;
 signal J,K,Q,notQ: std_logic_vector(0 to 2);
 signal JK0,JK1,JK2: std_logic_vector(0 to 1);
begin
  JK0 <=(J(0),K(0));JK1 <=(J(1),K(1));
  JK2 <=(J(2),K(2));
  process(Clk)
    begin
       if(Clk'event and Clk='0') then
            case JK0 is
                  when "00"   =>  Null;
                  when "01"     => Q(0)<='0';
                  when "10"     => Q(0)<='1';
                  when others => Q(0)<= not Q(0);
            end case;
            case JK1 is
                  when "00"   =>  Null;
                  when "01"     => Q(1)<='0';
                  when "10"     => Q(1)<='1';
                  when others => Q(1)<= not Q(1);
            end case;
            case JK2 is
                  when "00"   =>  Null;
                  when "01"     => Q(2)<='0';
                  when "10"     => Q(2)<='1';
                  when others => Q(2)<= not Q(2);
            end case;
       end if;
   end process;
   notQ<=not Q;
   J(0)<=Q(1) nand Q(2); K(0)<='1';
   J(1)<=Q(0);     K(1)<= notQ(0) nand notQ(2);
   J(2)<=Q(1) and Q(0); K(2)<=Q(1);
end Behavioral;
```

35. Đoạn mô tả nào mô tả đúng cho mạch giải mã BCD sang mã 7 segment.

A.

```
entity BCDto7seg is
Port( BCD:in
      std_logic_vector(3 downto
0);
       Seg : out
     std_logic_vector(6 downto
0));
end BCDto7seg;
architecture Beh of BCDto7seg is
begin
 with BCD select
        --abcdefg"
    Seg<= "1111110" when x"0",
          "0110000" when x"1",
          "1101101" when x"2",
          "1111001" when x"3",
          "0110011" when x"4",
          "1011011" when x"5",
          "1011111" when x"6",
          "1110000" when x"7",
          "1111111" when x"8",
          "1111011" when x"9",
          "0000000" when others;
end Beh;
```

C.

```
entity BCDto7seg is
Port ( BCD:in
      std_logic_vector(3 downto
0);
       Seg : out
     std_logic_vector(6 downto
0));
end BCDto7seg;
architecture Beh of BCDto7seg is
begin
 with BCD select
        --abcdefg"
    Seg<= "1111110" when  x"0",
          "0110000" when x"1",
          "1101101" when x"2",
          "1111001" when x"3",
          "0110011" when x"4",
          "1011011" when x"5",
          "1011111" when x"6",
          "1111111" when x"7",
          "1111111" when x"8",
          "1111111" when x"9",
          "0000000" when others;
end Beh;
```

B.	D.
entity BCDto7seg is Port (BCD:in std_logic_vector(3 downto 0); Seg : out std_logic_vector(6 downto 0)); end BCDto7seg; architecture Beh of BCDto7seg is begin with BCD select --abcdefg" Seg<= "1111110" when x"0", "0110000" when x"1", "1101101" when x"2", "1111001" when x"3", "0000000" when others; end Beh;	entity BCDto7seg is Port (BCD:in std_logic_vector(3 downto 0); Seg : out std_logic_vector(6 downto 0)); end BCDto7seg; architecture Beh of BCDto7seg is begin with BCD select --abcdefg" Seg<= "1011111" when x"6", "1110000" when x"7", "1111111" when x"8", "1111011" when x"9", "0000000" when others; end Beh;

36. Đoạn mô tả nào mô tả đúng cho mạch hợp kênh 8 vào – 1 ra:

A.

```
entity Mux is
end Mux;
architecture Behavioral of Mux
is
    signal I :
      std_logic_vector(8 downto
0);
    signal SEL:
      std_logic_vector(4 downto
0);
    signal Y :std_logic;
begin
    with SEL select
         --abcdefg"
    Y <=   I(0) when "0000",
          I(1) when "0001",
          I(2) when "0010",
          I(3) when "0011",
          I(4) when "0100",
          I(5) when "0101",
          I(6) when others;
end Behavioral;
```

C.

```
architecture Behavioral of Mux
is
    signal I :
      std_logic_vector(7 downto
0);
    signal SEL:
      std_logic_vector(2 downto
0);
    signal Y :std_logic;
begin
      process
      begin
         case SEL is
           when "000" =>
Y<=I(0);
           when "001" =>
Y<=I(1);
           when "010" =>
Y<=I(2);
           when "011" =>
Y<=I(3);
           when "100" =>
Y<=I(4);
           when "101" =>
Y<=I(5);
           when "110" =>
Y<=I(6);
           when others =>
Y<=I(7);
        end case;
      end process;
end Behavioral;
```

B.

```
entity Mux is
end Mux;
architecture Behavioral of Mux
is
    signal I :
      std_logic_vector(7 downto
0);
    signal SEL:
      std_logic_vector(2 downto
0);
    signal Y :std_logic;
begin
    with SEL select
         --abcdefg"
    Y <=  I(0) when "000",
          I(1) when "001",
          I(2) when "010",
          I(3) when "011",
          I(4) when "100",
          I(5) when "101",
          I(6) when "110",
          I(7) when others;
end Behavioral;
```

D.

```
architecture Behavioral of Mux
is
    signal I :
      std_logic_vector(7 downto
0);
    signal SEL:
      std_logic_vector(2 downto
0);
    signal Y :    std_logic;
begin
      process(I)
      begin
         case SEL is
           when "000" =>
Y<=I(0);
           when "001" =>
Y<=I(1);
           when "010" =>
Y<=I(2);
           when "011" =>
Y<=I(3);
           when "100" =>
Y<=I(4);
           when "101" =>
Y<=I(5);
           when "110" =>
Y<=I(6);
           when others =>
Y<=I(7);
        end case;
      end process;
end Behavioral;
```

ĐÁP ÁN VÀ HƯỚNG DẪN TRẢ LỜI

CHƯƠNG 1

1. Bit là số nhị phân có một chữ số. 1byte = 8 bit.
2. b
3. c
4. a
5. d
6. a

CHƯƠNG 2

Bài 1.

1. a
2. b

Bài 2.2

1. c
2. b

Bài 2.3

d

Bài 2.4

d. Do đều bằng A+AB

Bài 2.5

- Mức logic và phân tích
- Trễ truyền lan và phân tích
- Công suất tiêu thụ và phân tích
- Hệ số ghép tải và phân tích
- Độ phòng vệ nhiễu và phân tích
- Một số tham số khác

Bài 2.6

c

Bài 2.7

c

Bài 2.8

- Nêu được khái niệm về tối ưu hoá mạch điện các họ cổng

- Công cụ tối ưu hoá
- Đưa ra ví dụ và phân tích hiệu quả kỹ thuật, kinh tế của việc tối ưu hoá

Bài 2.10

a

Bài 2.11

d

Bài 2.12

c

CHƯƠNG 3

1.d	**2.a**
3.d	**4.b**
5.c	**6.a**
7.b	**8.c**
9.d	**10.b**
11.a	**12.d**
13.d	**14.a**

CHƯƠNG 4

1.a	**2.d**
3.c	**4.c**
5.c	**6.d**
7.b	**8.c**
9.a	**10.c**
11.a	**12.d**
13.c	**14.a**
15.b	**16.b**
17.a	**18.b**
19.c	**20.d**

CHƯƠNG 5

1.a	**2.c**

3.c **4.b**

5.d **6.a**

7.c **8.d**

9.d **10.c**

11.a **12.b**

13.d **14.c**

15.c **16.a**

17.d **18.b**

19.a **20.a**

21.b **22.d**

23.b **24.a**

25.b **26.c**

27.c **28.d**

29.c **30.a**

31.b **32.d**

33.c **34.a**

35.c **36. Xem ví dụ phần 5.4.1.2**

37. Xem ví dụ phần 5.4.1.2 **38.d**

39.b **40.a**

CHƯƠNG 6

1.c **2.a**

3.b **4.d**

5.b **6.b**

7.c **8.c**

9.a **10.d**

CHƯƠNG 7

1.a **2.c**

3.c **4.b**

5.c **6.a**

7.b	**8.a**
9.c	**10.c**

CHƯƠNG 8 VÀ CHƯƠNG 9

1.C	**2.D**
3.C	**4.D**
5.C	**6.D**
7.B	**8.D**
9.D	**10.C**
11.B	**12.B**
13.D	**14.D**
15.B	**16.A**
17.C	**18.D**
19.A	**20.D**
21.D	**22.A**
23.C	**24.B**
25.D	**26.A**
27.B	**28.C**
29.C	**30.C**
31.B	**32.A**
33.C	**34.D**
35.A	**36.B**

TÀI LIỆU THAM KHẢO

1. *Giáo trình Kỹ thuật số* - Trần Văn Minh, NXB Bưu điện 2002.
2. *Cơ sở kỹ thuật điện tử số*, Đại học Thanh Hoa, Bắc Kinh, NXB Giáo dục 1996 .
3. *Kỹ thuật số*, Nguyễn Thúy Vân, NXB Khoa học và kỹ thuật 1994.
4. *Toán logic và kỹ thuật số,* Nguyễn Nam Quân - Khoa ĐHTC xuất bản - 1974
5. *Lý thuyết mạch logic và Kỹ thuật số*, Nguyễn Xuân Quỳnh - NXB Bưu điện - 1984
6. *Fundamentals of logic design*, fourth edition, Charles H. Roth, Prentice Hall 1991.
7. *Digital engineering design*, Richard F.Tinder, Prentice Hall 1991 .
8. *Digital design principles and practices*, John F.Wakerly, Prentice Hall 1990.
9. *VHDL for Programmable Logic* by Kevin Skahill, Addison Wesley, 1996
10. *The Designer's Guide to VHDL* by Peter Ashenden, Morgan Kaufmann, 1996.
11. *Analysis and Design of Digital Systems with VHDL* by Dewey A., PWS Publishing, 1993.



MỤC LỤC

ĐIỆN TỬ SỐ

Mã số : 497DTS210

Chịu trách nhiệm bản thảo

TRUNG TÂM ĐÀO TẠO BƯU CHÍNH VIỄN THÔNG 1